jamk.fi

Trac-projektin siirto GitLabiin

Toni Pudas

Opinnäytetyö
Toukokuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Pudas, Toni	Opinnäytetyö, AMK	Kesäkuu 2018
	Sivumäärä 51	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Trac-projektin siirto GitLabiin

Tutkinto-ohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Marko Rintamäki

Toimeksiantaja(t)
Esa Salmikangas

Tiivistelmä

Työssä pyrittiin etsimään korvaaja Betty-projektin hallintaan käytetylle Trac-järjestelmälle, jota ylläpidettiin JAMK:n palvelimilla.

Työssä vertailtiin erilaisia Trac-järjestelmää vastaavia järjestelmiä, esiteltiin vertailun tulokset toimeksiantajalle ja siirrettiin Betty-projekti hänen valitsemaansa järjestelmään. Toimeksiantaja määritteli myös tiedot, jotka hän halusi siirrettävän valittuun järjestelmään eli koodirepositoriot ja wiki-sivut. Issuet jätettiin siirto-operaation ulkopuolelle. Työssä käytettiin parhaiksi katsottuja siirtotapoja edellä mainituille tiedoille.

Wiki-sivut siirrettiin lataamalla niiden kaikki versiot .NETin http-kirjastoja käyttäen ja muuttamalla ne Tracin käyttämästä Wiki markup -muodosta GitLabin käyttämään Markdown-muotoon. Tämän jälkeen wiki-sivut siirrettiin GitLab-projektin wiki-sivuille suunnattuun Git-repositorioon. Trac-projektin Subversion-repositorio muutettiin Git-repositorioksi git svn -työkalulla ja sitä peilattiin luodun GitLab-projektin koodirepositorioon.

Toimeksiantaja oli tyytyväinen migraation tuloksiin, vaikka osa wiki-sivujen muotoilutiedoista hävisi siirron aikana, eikä siirto-operaation onnistumista tarkistettu ohjelmallisesti vaan manuaalisesti. Myös osa Subversion-repositorion ominaisuuksista jäi siirtämättä, mutta ne eivät ole oleellisia jatkokehityksen kannalta.

Avainsanat (asiasanat)
ALM, konfiguraationhallinta, versionhallinta, Subversion, Git, Trac

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Pudas, Toni	Bachelor's thesis	June 2018
		Language of publication: Finnish
	Number of pages: 51	Permission for web publication: x

Title of publication
Migrating Trac project to GitLab

Degree programme
Software Engineering

Supervisor(s)
Marko Rintamäki

Assigned by
Esa Salmikangas

Abstract

This assignment aimed to find a substitute for Trac system used to manage the Betty project. The system was maintained on JAMK servers.

Different systems similar to Trac were compared, the results of the comparison were introduced to the assignor and the Betty project was migrated to the chosen system. The assignor defined the data he wanted to migrate to the chosen system: the code repository and the wiki pages. Issues were left outside of migration. The best ways to migrate were chosen and used in this assignment.

The wiki pages were migrated by downloading all their all versions with .NET http libraries and by converting them from Wiki markup syntax used by Trac to Markdown syntax used by GitLab. After that, the wiki pages were migrated to the Git repository designated for wiki pages in the GitLab project. The Subversion repository of the Betty project was converted to a Git repository with the git svn tool and was mirrored to the code repository of the created GitLab project.

The assignor was pleased with the results of the migration although some of the data regarding the appearance of wiki pages was lost and the success of the migration of wiki pages was not confirmed programmatically but manually. Additionally, some of the properties of the Subversion repository were not transferred; however, they are not of relevance for further development.

Keywords/tags (subjects)
ALM, configuration management, version management, Subversion, Git, Trac

Miscellaneous

Sisältö

Kuviot

1 Johdanto

1.1 Toimeksiantaja: Jyväskylän Ammattikorkeakoulu Oy

Jyväskylän Ammattikorkeakoulu Oy on perustettu vuonna 1994, ja yhtiön tarkoitus on ylläpitää ammattikorkeakoulua. Omistajia ovat Jyväskylän kaupunki, Äänekosken ammatillisen koulutuksen kuntayhtymä POKE sekä Jämsän kaupunki. (JAMKin hallinto n.d.) Yhtiön omistamassa ammattikorkeakoulussa on yli 8000 opiskelijaa ja henkilöstön edustajia noin 600 (Jyväskylän ammattikorkeakoulu n.d.).

1.2 Tavoitteet

Tämän työn pääasiallinen tavoite oli Jyväskylän ammattikorkeakoulun Labranet-ympäristön Trac-palvelimella sijaitsevan projektin siirtäminen niin sanotusti modernimpaan projektinhallintaympäristöön. Kyseiseen projektiin sisältyy Jyväskylän ammattikorkeakoulun rakennuslaboratoriossa käytetty Betty-ohjelmisto, ja projektia siihen liittyvine repositorioineen ja wikisivuineen on hallittu edellä mainitun Trac-palvelimen avulla.

Tilaajana toiminut JAMKin lehtori, joka myös ehdotti opinnäytetyön aihetta, kertoi aiheen keksimisen pääasialliseksi motiiviksi sen, etteivät Labranetin ylläpitäjät ole aktiivisesti päivittäneet Trac-palvelinta. Hän myös ehdotti opinnäytetyön osa-alueiksi uuden projektinhallintaympäristön valinnan, erilaiset tiedonsiirtometodien tutkimisen sekä tiedon eheyden varmistamisen projektin siirtämisen yhteydessä.

2 Ohjelmistokehitys

2.1 Prosessivaiheet ohjelmistokehityksessä

2.1.1 Yleistä

Ohjelmistokehitysprosessi on joukko toisiinsa liittyviä aktiviteetteja, jotka johtavat ohjelmiston valmistumiseen. Erilaisia prosessimalleja on useita, esimerkiksi suunnitteluun vahvasti nojautuva vesiputousmalli, jossa kehitysprosessi on jaettu eri vaihei-

siin, jotka suoritetaan ennalta määrätyssä järjestyksessä. Vaikka erilaisia kehitysprosesseja ja prosessimalleja on lukuisia, ne sisältävät kuitenkin yhteisiä, kaikkeen ohjelmistokehitykseen kuuluvia aktiviteetteja. Nämä aktiviteetit ovat ohjelmiston määrittely, kehittäminen, validointi ja evoluutio. (Sommerville 2016, 44-45.)

2.1.2 Ohjelmiston määrittely

Ohjelmiston määrittelyllä pyritään ymmärtämään ja määrittelemään järjestelmältä vaaditut palvelut ja tunnistamaan järjestelmän toimintaa ja kehitystä rajoittavat tekijät. Se on kriittinen vaihe ohjelmiston kehityksessä, sillä huonosti tehty määrittely johtaa ongelmiin kehityksen myöhemmissä vaiheissa. Ohjelmiston määrittely koostuu kolmesta pääaktiviteetista: vaatimusten aikaansaamisesta ja niiden analysoinnista, vaatimusmäärittelystä sekä vaatimusten validoinnista. (Sommerville 2016, 54-55.) Näiden aktiviteettien lähempi tarkastelu ei kuitenkaan ole oleellista tämän opinnäytetyön kannalta.

2.1.3 Ohjelmiston kehittäminen eli suunnittelu ja toteuttaminen

Ohjelmiston toteuttaminen on suoritettavan järjestelmän luomiseen asiakkaalle tähtäävä prosessi. Se voi koostua ohjelmiston suunnittelusta ja ohjelmoinnista, mikäli ei käytetä ketteriä menetelmiä, joissa suunnittelu toteutetaan epävirallisemmin. Suunnittelun tarkoituksena on luoda kuvaus ohjelmiston rakenteesta, järjestelmän käyttämistä tietomalleista ja rakenteista, komponenttien välisistä rajapinnoista ja toisinaan myös käytetyistä algoritmeista. Ohjelmiston suhteet muihin sen lopullisessa suoritusympäristössä oleviin ohjelmistojärjestelmiin, kuten esimerkiksi käyttöjärjestelmään ja tietokantaan, selvitetään suunnitteluvaiheessa. Vaiheen aktiviteetit voivat olla esimerkiksi arkkitehtuuri-, tietokanta-, rajapinta- ja komponenttisuunnittelu sekä komponenttien valitseminen. Nämä aktiviteetit johtavat suunnittelutuotoksiin, joiden perusteella voidaan aloittaa ohjelmiston toteutus, vaikkakin suunnittelu- ja toteutusvaiheet ovat monissa tapauksissa päällekkäisiä. Mitä varsinaiseen ohjelmointiin tulee, ei sille ole mitään yleispätevää prosessia, mutta se sisältää yleensä testausta ja virheenkorjausta. (Sommerville 2016, 57-58.)

2.1.4 Ohjelmiston validointi

Ohjelmiston validointi, tai verifikointi ja validointi, on aktiviteetti, jonka tarkoituksena on osoittaa, että ohjelmisto on sen määritelmän ja asiakkaan odotusten mukainen. Validointi sisältää ohjelmiston tarkastelua ja katselmointia kehityksen eri vaiheissa, mutta pääasiassa se on ohjelmiston testausta simuloitua testitietoa apuna käyttäen. Ohjelmistot testataan kokonaisuutena, mutta suuremmissa ohjelmistoissa testataan vielä yksittäiset komponentit erikseen, yleensä jo kehitysvaiheen aikana. Sovelluksille ja räätälöidyille ohjelmistoille tehdään lisäksi käyttäjätestausta ja sovellusten kohdalla tätä prosessia kutsutaan joskus myös beta-testaukseksi. Tarkemmat testauskäytännöt määrittyvät kehityksessä käytetyn prosessimallin mukaisesti. (Sommerville 2016, 58-60.)

2.1.5 Ohjelmiston evoluutio

Ohjelmistojen joustavuuden saavuttamiseksi ohjelmistoja sisällytetään yhä enenevissä määrin suuriin ja monimutkaisiin järjestelmiin. Tämä joustavuuden tavoittelu on kuitenkin edullisempaa kuin laitteistojen uusiminen järjestelmien toimimattomuuden vuoksi, sillä järjestelmät ovat muokattavissa vielä niiden kehityksen jälkeenkin. Perinteisesti ohjelmiston kehitys ja sen evoluutio eli ohjelmiston ylläpito on nähty hyvin erilasina prosesseina, mutta tämä näkemys on nykyään virheellinen, sillä todella harvat järjestelmät ovat täysin uusia. Ohjelmiston kehitys tulisikin nähdä koko sen elinkaaren kestävänä evoluutioprosessina, minkä aikana se muovautuu koko ajan muuttuvien vaatimusten ja asiakkaan tarpeiden mukaisesti. (Sommerville 2016, 60-61.)

Evoluutioon liittyy läheisesti myös muutostenhallinta. Muutoksia tapahtuu kaikissa suurissa ohjelmistoprojekteissa yritysten liiketoiminnallisten muutosten yhteydessä ja uusien teknologioiden ilmestyessä. Muutokset lisäävät ohjelmistokehityksen kustannuksia, mutta nämä kustannukset ovat minimoitavissa eri lähestymistavoilla, joilla pyritään minimoimaan muutoksien tekemiseen tarvittava työmäärä.

2.2 Projektinhallinta

Eräs ammattimaisen ohjelmistokehityksen oleellisimmista tavoitteista on luoda asiakkaan odotuksia vastaava ohjelmisto rajoitettujen resurssien, ajan ja rahan, puitteissa.

Vajaiden resurssien vuoksi on erityisen tärkeää varmistaa kehitystiimin yhtenäisyys ja toimintakyky. Ohjelmistoprojektien hallinnasta tekee erityisen haasteellista muunlaisten projektien hallintaan verrattuna muun muassa tuotteen aineettomuus, suurten projektien vahva yksilöllisyys sekä ohjelmistokehitysprosessien erilaisuus. Viimeksi mainittu vaikuttaa oleellisesti projektinhallintapoihin ohjelmistoa kehittävän yrityksen koon ja sen yhteisökulttuurin ohella. Muita vaikuttavia tekijöitä ovat ohjelmiston koko ja tyyppi sekä asiakas, joka voi olla yrityksen sisäinen tai ulkoinen toimija. (Sommerville 2016, 642-643.)

3 ALM

3.1 Taustaa

Liiketoiminta on muuttunut globaaliksi, ja yrityksillä on sovelluskehitystiimejä ympäri maailmaa. Tämä luo haasteita tiimien väliseen yhteistyöhön, esimerkiksi vaatimusten ja lähdekoodin hallinnan suhteen. (Rossberg, Ehn & Olausson 2014, 1.)

Yritysten järjestelmäkehityksessä on kolme pääosaa: prosessit, liiketoimintasäännöt ja tieto. Prosesseja käytetään yritysten liiketoiminnan tukemiseksi, ja niihin voivat sisältyä muun muassa Scrumin, XP:n ja RUP:n prosessit. Liiketoimintasäännöt taas määrittelevät, mitä yrityksessä pitää, saa ja ei saa tehdä. Tieto-käsitteeseen sisältyy kaikki yritykseen liittyvä tieto, esimerkiksi asiakastiedot ja tilaustiedot. (Rossberg, Ehn & Olausson 2014, 3.)

3.2 ALM

ALM määrittelee koko ohjelmistokehitysprosessin kiertokulun (Rossberg, Ehn & Olausson 2014, 1).

Se on yksi niistä prosesseista, joita käytetään yrityksen tarpeiden ja vaatimusten tyydyttämiseksi siten, että saadaan lisättyä yrityksen liikearvoa sen luoman ohjelmiston kautta. ALM-prosessin väärinkäytöstä puolestaan seuraa liikearvon pieneneminen. (Rossberg, Ehn & Olausson 2014, 3.)

Nykypäivän liiketoimintaympäristössä ALM-prosessin hallinta on tärkeää, sillä se edistää yhteistyötä fyysisesti eri paikoissa sijaitsevien kehitystiimien välillä. Yrityksen kilpailuedun ylläpito edellyttää prosessin automatisointia ja hienosäätöä, sekä ketteryyttä, mikä voidaan saavuttaa ALM-prosessin tukemilla järjestelmän toteutus- ja suunnittelutapoihin tehdyillä muutoksilla. (Rossberg, Ehn & Olausson 2014, 4.)

ALM-työkalujen on tarkoitus edesauttaa yrityksen liiketoimintaprosesseissa tapahtuvien muutoksien heijastumista yrityksen ohjelmistokehitykseen, mutta kyseiset työkalut ovat vain osa ALM-prosessia (Rossberg, Ehn & Olausson 2014, 5). Markkinoilla on muun muassa Microsoftin, HP:n, IBM:n sekä Atlassianin tarjoamia työkaluja, jotka voidaan luokitella ALM-alustoiksi. Näiden työkalujen avulla voidaan muodostaa silta yritysten liiketoiminta- ja It-yksiköiden välille. (Rossberg, Ehn & Olausson 2014, 2.)

3.3 Projektin ennalta-arvaamattomuus

Projektin onnistumista mitataan yleensä kolmella kriteerillä: projektin suoritus ajallaan, projektin suoritus määrättyjen resurssien puitteissa sekä projektin tavoitteiden saavuttaminen (Rossberg, Ehn & Olausson 2014, 5).

Ohjelmistoprojektit ovat siinä mielessä epävarmoja, että niihin liittyy runsaasti asioita, jotka voivat muuttua projektin aikana. Näin ollen aikataulua on vaikea arvioida alussa, joten aikaa ei tule pitää kaikkein oleellisimpana projektin tilanteen indikaattorina. (Rossberg, Ehn & Olausson 2014, 6.)

Epävarmuutta projekteihin lisää myös se, ettei kehitettävien järjestelmien monimutkaisuutta eikä järjestelmään myöhemmin tehtäviä muutoksia pystytä etukäteen ennustamaan. Tämä lisää kehitykseen käytettävää aikaa, ja näin ollen myös projektin kokonaiskustannuksia. (Rossberg, Ehn & Olausson 2014, 6.)

Projektin vaatimuksiin tehdyt muutokset vaikuttavat myös projektin tavoitteiden saavutettavuuteen. Se, että projekti on saavuttanut alkuperäiset tavoitteensa, ei välttämättä hyödytä asiakasta, jonka tarpeet ovat muuttuneet projektin aikana. Näin ollen tulisi ennemminkin keskittyä asiakkaalle valmiin tuotteen kautta toimitettavaan liikearvoon. Tässä auttaa hyvän kehitys- ja ALM-prosessin käyttö. (Rossberg, Ehn & Olausson 2014, 6.)

3.4 Projektin onnistumiseen vaikuttavat tekijät

3.4.1 Yrityksen IT- ja liiketalousosastojen yhteisymmärrys

Yrityksen IT- ja liiketalousosastojen välisen ymmärryksen tasoon vaikuttaa se, kuinka vahvasti osapuolet käsittelevät projektia oman osaamisalueensa perspektiivistä, ja tämä taso vaikuttaa osapuolten väliseen kommunikaatioon, ja tätä kautta projektin onnistumiseen. (Rossberg, Ehn & Olausson 2014, 7.)

3.4.2 Kehitysprosessi

Kehitysprosessin käyttö tai sen puuttuminen vaikuttaa projektin onnistumiseen. Kehitysprosessi ei silti yksinään takaa onnistumista, vaan myös kehityksessä käytettävien työkalujen tulisi tukea sitä saumattomasti. Pakotettu ja epäkäytännöllinen prosessi haittaa työskentelyä, ja voi johtaa sen epätäydelliseen toteutumiseen. Prosessin tulisi myös olla joustava projektiin tehtävien muutosten helpottamiseksi. (Rossberg, Ehn & Olausson 2014, 7.)

3.4.3 Maantieteellinen jakauma

Kehitystyökalujen suuri määrä johtaa siihen, että ajantasaisen tiedon ylläpito kaikissa järjestelmissä sekä kehitysprosessin soveltaminen kaikki työkalut huomioon ottaen on hyvin työlästä. Tästä syystä markkinoilla on työkalupaketteja, jotka vähentävät kehittäjän tarvetta siirtyä työkalusta toiseen, ja joissakin jopa tiedon synkronisointi eri työkalujen välillä tapahtuu automaattisesti. (Rossberg, Ehn & Olausson 2014, 8.)

3.4.4 Työkalujen synkronointi

Pysyminen ajan tasalla projektien suhteen on suuri haaste yrityksille, ja resurssien hyvä hallinta on tärkeää projektien sujuvuuden kannalta. PPM (Project Portfolio Management) mahdollistaa paremman resurssien ohjaamisen projekteille. Forresterin mukaan PPM antaa tietoon perustuvan prosessin projektien vertailuun, priorisointiin ja tarkkailuun. Se yhdistää strategiseen suunnittelun, resurssien ja budjetin ohjaamisen, projektin valinnan ja toteutuksen sekä valmiin projektin mittauksen prosessit. (Rossberg, Ehn & Olausson 2014, 9.)

3.4.5 Projektin koko

Mitä suurempi projekti on, sitä suurempi on yksittäisten kehittäjien määrä, ja sitä pitkäkestoisempi se on. Nämä ominaisuudet yleensä vaikeuttavat projektin hallintaa, varsinkin jos kehitystiimit ovat maantieteellisesti hajallaan. (Rossberg, Ehn & Olausson 2014, 9.)

3.4.6 Muita edellytyksiä

Standishin mukaan kymmenen tärkeintä projektin onnistumisen edellytyksiä ovat:

- Loppukäyttäjien osallistuminen
- Johdon tuki
- Selkeät liiketoimintatavoitteet
- Projektin laajuuden optimointi
- Ketterä prosessi
- Projektipäällikön asiantuntemus
- Taloudellinen hallinta
- Virallinen metodologia
- Standardittyökalut ja –infrastruktuuri

Suurin osa näistä edellytyksistä on saavutettavissa hyvän ALM-prosessin ja hyvän integroidun ALM-työkaluston avulla. (Rossberg, Ehn & Olausson 2014, 11.)

3.5 ALM-prosessi

3.5.1 Prosessi lyhyesti

ALM-prosessi mahdollistaa paremman kontrollin projektin lopputuloksen suhteen (s. 11) Se alkaa liiketoiminnallisesta tarpeesta lähtevästä ohjelmiston synnystä. Alkuvaiheessa toteutetaan usein portfolion järkeistäminen, mikä tarkoittaa uuden ohjelmiston kehittämisen tarpeen selittämistä. Mikäli tarve on olemassa, tapahtuu siirtyminen ohjelmistokehityksen elinkaareen, johon sisältyy ohjelmiston kehitys, testaus ja käyttöönotto. Tämän jälkeen ohjelmisto siirtyy toimintavaiheeseen, jossa sitä ylläpidetään ja parannellaan. Toimintavaiheessa olevaa ohjelmistoa päivitetään ja siinä ilmeneviä virheitä korjataan muutospyyntöjen (CR) perusteella, jolloin prosessi alkaa taas alusta. (Rossberg, Ehn & Olausson 2014, 13.)

3.5.2 ALM-prosessin roolit

ALM-prosessin rooleihin kuuluvat ainakin:

- Osakkaat
- Liiketoimintapäällikkö
- Projektipäällikkö, projektin omistaja tai SCRUM-mestari
- Projektinhallintatoimiston päätöksentekijät
- Liiketoiminta-analyytikko
- Järjestelmäarkkitehti
- UX-suunnittelutiimi
- Tietokantojen ylläpitäjät
- Kehittäjät
- Testaajat
- Ohjelmiston toiminnasta ja ylläpidosta vastaava henkilöstö

Ketterä kehitys on erityistapaus siinä mielessä, että siihen kuuluu vain kolme roolia; tuotteen omistaja, SCRUM-mestari ja kehitystiimin jäsenet. ALM-prosessin roolit kuitenkin sisältyvät näihin rooleihin. (Rossberg, Ehn & Olausson 2014, 15.)

3.6 ALM-prosessin eri näkökulmat

ALM-prosessiin voidaan ottaa neljä eri näkökulmaa; ohjelmistokehityksen elinkaari, palvelun hallinta tai operointi, ohjelmiston portfolion hallinta sekä edellä mainitut näkökulmat yhdistävä näkökulma. (Rossberg, Ehn & Olausson 2014, 16.)

3.6.1 Ohjelmistokehityksen elinkaari

Ohjelmistokehityksen elinkaaren näkökulmasta ALM-prosessi varmistaa, että on olemassa kaikki ohjelmistokehityksen elinkaaren aktiviteetit kattavat prosessit. Se myös varmistaa kehityksen aikaisissa aktiviteeteissa käytettyjen tai niissä syntyneiden artefaktien jäljitettävyyden sekä kehitystyön kokonaisvaltaisen edistymisen raportoinnin. (Rossberg, Ehn & Olausson 2014, 17.) ALM-prosessin tarkoitus on pitää elinkaareen kuuluvat aktiviteetit synkronisoituina. Näin saadaan nopeammin selville tehottomat aktiviteetit. (Rossberg, Ehn & Olausson 2014, 18.)

3.6.2 Palvelunhallinta tai operointi

OGC:n mukaan ALM keskittyy aktiviteetteihin jotka liittyvät ohjelmiston käyttöönottoon, operointiin, tukeen sekä optimointiin, ja sen päätavoite on varmistaa, että

koonnin ja käyttöönoton jälkeen ohjelmisto täyttää sille määritellyn palvelutason. (Rossberg, Ehn & Olausson 2014, 18.)

3.6.3 Ohjelmistoportfoliohallinta

Ohjelmistoportfoliohallinta on osa projektiportfoliohallintaa. Projektiportfoliohallintaosasto hallitsee resursseja ja työstettäviä projekteja. Ohjelmistoportfoliohallinnan näkökulmassa keskitytään ohjelmistoon tuotteena, ja tuotteen kiertokulku alkaa aina liiketoimintasuunnitelmasta. Liiketoimintasuunnitelman kautta tarkasteltavaksi päätyvät sekä uudet tuotteet, että ylläpitovaiheessa olevat tuotteet, joiden liiketoimintavaatimukset ovat muuttuneet tai jotka vaativat uutta julkaisua. (Rossberg, Ehn & Olausson 2014, 18.)

3.6.4 Yhtenäinen näkökulma

Yhtenäisessä näkökulmassa yhdistetään edellä mainitut näkökulmat liiketoiminnan näkökulmaan, ja tässä näkökulmassa tarkastellaan ohjelmiston koko elinkaarta. (Rossberg, Ehn & Olausson 2014, 19.)

3.7 ALM-prosessin kolme tukipilaria

3.7.1 Jäljitettävyys

Vaatimusten on oltava nähtävillä tuotteessa, esimerkiksi suunnittelumalleissa ja yksikkötesteissä. Näin mahdollistetaan vaatimusten täyttämisen todistaminen, ja helpotetaan ohjelmiston päivittämistä ja bugien korjaamista. Vaatimusten jäljitettävyys voi olla myös ehdoton vaatimus jonkin ulkopuolisen tahon, kuten viranomaisen, toimesta. (Rossberg, Ehn & Olausson 2014, 20.)

3.7.2 Korkean tason prosessien automatisointi

Muun muassa hyväksymisprosessit ohjelmiston kehityksen eri aktiviteettien välillä voidaan hoitaa automaattisesti. Prosessien automatisointi säästää aikaa ja pienentää virheiden määrää manuaaliseen prosessiin verrattuna. (Rossberg, Ehn & Olausson 2014, 20.)

3.7.3 Kehityksen edistymisen näkyvyys

Kehitysprojektien sidosryhmillä on usein rajattu näkymä projektin edistymiseen. Tilanneraporttien tekeminen vie paljon aikaa ja vaivaa, mutta projektin tilanteen näkyvyys sidosryhmille on tärkeää tehokkaan ALM-prosessin kannalta. (Rossberg, Ehn & Olausson 2014, 20.)

3.8 ALM-työkalut

3.8.1 ALM-työkalujen lyhyt historia

Edellisessä luvussa mainitut ALM:in tukipilarit voi toteuttaa manuaalisesti, esimerkiksi taulukkolaskentaohjelmaa ja muistilappuja käyttäen, tai automaattisesti. Automaattinen toteutus tapahtuu ALM-työkalujen avulla. Niitä käyttämällä voidaan varmistaa jäljitettävyyden, raportoinnin sekä monien muiden aktiviteettien toteutuminen. Lisäksi niiden käyttö tehostaa työn tuottavuutta saavutetun ajansäästön myötä ja pienentää virheiden sattumisen riskiä. (Rossberg, Ehn & Olausson 2014, 20.)

Serena Software Inc. on yksi ALM-työkaluja tuottava yritys, jonka mukaan on olemassa kahdeksan ALM-käsitettä. Nämä käsitteet ovat (Rossberg, Ehn & Olausson 2014, 20):

- Ohjelmiston mallinnus
- Issueiden hallinta
- Suunnittelu
- Konstruktio
- Tuotannon valvonta
- Koonti
- Testaus
- Julkaisun hallinta.

Yrityksen mukaan näiden asioiden synkronointiin työkaluja, jotka auttavat seuraavien aktiviteettien yksinkertaistamisessa (Rossberg, Ehn & Olausson 2014, 20):

- Raportointi
- Jäljitettävyys
- Toimintaperiaatteet
- Menettelytavat
- Prosessit
- Yhteistyö

Butler Groupin Michael Azhoffin mukaan epäonnistuneiden projektien väheneminen johtuu merkittävistä muutoksista ohjelmistokehityksessä: avoimen lähdekoodin projekteista, ketterän kehityksen ympärille kerääntyneestä yhteisöstä sekä työkalujen, varsinkin ALM-työkalujen, kehityksestä. (Rossberg, Ehn & Olausson 2014, 22.)

3.8.2 ALM 1.0

Ohjelmistojen monimutkaistuminen on lisännyt ryhmäytymistä erityisosaamisalueiden mukaan IT-organisaatioissa. Tämä ryhmäytyminen vaatii kuitenkin yhteistyötä eri ryhmien välillä, jotta projektit onnistuvat. ALM-työkalut ovat olleet roolikeskeisiä, eivätkä eri rooleille suunnatut työkalut ole olleet vuorovaikutuksessa keskenään. ALM-prosessi on synkronoitava, jotta roolikeskeiset prosessit ovat osa kokonaisprosessia. (Rossberg, Ehn & Olausson 2014, 22.)

Eri rooleille suunnattujen työkalujen välillä ei tämän ALM-version aikana ollut täyttä integraatiota, vaan pisteestä-pisteeseen-integraatio. Tässä integraation muodossa eri työkalut kommunikoivat toistensa kanssa hieman, mutta niillä kullakin on omat erilliset datarepositorionsa. (Rossberg, Ehn & Olausson 2014, 22.) Pisteestä-pisteeseen-integraatio johtaa siihen, että yhteys työkalujen välillä on hyvin heikko, ja voi katketa esimerkiksi silloin, kun jokin työkaluista päivitetään (Rossberg, Ehn & Olausson 2014, 23).

Pisteestä-pisteeseen-integraatio myös hankaloittaa työskentelyä tilanteessa, jossa sama projektin jäsen hoitaa kahden tai useamman roolin tehtäviä. Kun esimerkiksi kehittäjä testaa koodiaan, hän joutuu käyttämään kehitys- ja testaustyökaluja vuorotellen, ja tämä työkalujen vaihtelu vie aikaa varsinaiselta työtehtävältä. (Rossberg, Ehn & Olausson 2014, 23.) Lisäksi tässä esimerkkitapauksessa joudutaan yksittäiselle projektin jäsenelle hankkimaan lisenssit moneen eri työkaluun. Työkalujen suuri määrä johtaa siihen, että kokonaisuus on monimutkainen ja ylläpitokustannukset ovat suuret. (Rossberg, Ehn & Olausson 2014, 24.)

Perinteisissä ALM-työkaluissa on myös se ongelma, että ne voivat sisältää kohderoolin aktiviteetteja tukevien ominaisuuksien lisäksi myös joillekin muille rooleille suunnattuja ominaisuuksia, mutta kuitenkin niin rajoitetusti, että tarve erillisille työkaluille on yhä olemassa (Rossberg, Ehn & Olausson 2014, 24).

3.8.3 ALM 2.0

ALM 2.0 on pyrkimys yhdistää ALM-työkalut yhdeksi alustaksi, jonka tarkoitus on mahdollistaa kehitysaktiviteettien koordinointi ja hallinta. (s. 24) Sen tunnusmerkkeihin kuuluvat lisäosista rakennetut työkalut, työkalujen jakamat yhteiset palvelut, tietokantaneutraalius, avoimien integraatiostandardien käyttö sekä ulkoistetun workflown ohjaamat mikro- ja makroprosessit. (Rossberg, Ehn & Olausson 2014, 24.)

ALM 2.0 tarkoittaa siis helpompaa työkalujen käyttöä; se tarjoaa työkalun, joka tukee kolmannen osapuolen lisäosia ja joka on helppo integroida muihin järjestelmiin. (s. 25) Kehitystyö helpottuu myös, kun aiemmin erillisten työkalujen ominaisuuksia yhdistetään yhteen alustaan; näitä ominaisuuksia ovat muun muassa yhteistyö, workflow, turvallisuus sekä raportointi ja analysointi. Lisäksi tuki eri repositorioille on laaja. (Rossberg, Ehn & Olausson 2014, 26.)

Tähän ALM-versioon kuuluu myös prosessituki, eli ALM-prosessi on sisäänrakennettu työkaluihin, jotka tukevat lisäksi eri kehitysprosessien automatisointia. Kehitysorganisaatioiden omat ALM-prosessit voi implementoida työkaluihin esimerkiksi XML-muodossa, jolloin prosessitiedosto on heleposti muokattavissa, versioitavissa ja raportoitavissa. (Rossberg, Ehn & Olausson 2014, 27.)

3.8.4 ALM 2.0+

Suurin osa merkittävimmistä ALM-työkalujen valmistajista ei ole ottanut kaikkia ALM 2.0-version ominaisuuksia käyttöön. Tämä johtuu siitä, että ohjelmistoyrityksien on vaikea muuttaa juurtuneita toimintatapojaan esimerkiksi kehitysprosessien suhteen. Esimerkiksi Microsoftin työkaluja käyttäville yrityksille siirtyminen ALM:in 2.0-versioon ei ole vaikeaa ainakaan ohjelmoijille, sillä he käyttävät jo esimerkiksi Visual Studiota ja SharePointia. Microsoftin tuotteet eivät kuitenkaan ole kehittyneet repositorioneutraaliuden suhteen, vaan työkalut tukeutuvat yhä SQL Server -repositorioihin. (Rossberg, Ehn & Olausson 2014, 27.)

Ketterien menetelmien suosion kasvu viime vuosien aikana on myös muuttanut tapaa, jolla ALM:in tulee tukea kehittäjäryhmiä ja -organisaatioita. Työn ohjautuvuudessa on tapahtunut selkeä siirtyminen vaatimusmäärittelystä backlogiin, ja tämä

edellyttää myös työkalukokonaisuuksien tukea. ALM-työkalujen tuki ketterille menetelmille, kuten koonti-testaus -automaatiolle on tullut oleelliseksi. Testiajetun kehityksen yleistyessä kehittäjät tarvitsevat sitä tukevia työkaluja. Mikäli työkalu ei tue tätä kehitysprosessia, ei siitä ole juurikaan hyötyä prosessia hyödyntäville yrityksille. (s. 28) Yksi esimerkki ketteriä menetelmiä tukevasta työkaluista on Microsoftin Team Foundation Server. (Rossberg, Ehn & Olausson 2014, 28.)

On tapahtunut siirtyminen perinteisestä projektinhallinnasta kohti ketterää näkymää, jossa myös tuotteen omistaja ja Scrum-mestari tarvitsevat työkalujen tukea. Näille rooleille tärkeät aktiviteetit, eli backlogin siistiminen, ketterä arviointi ja suunnittelu sekä raportointi, tulee myös integroida ALM-ratkaisukokonaisuuteen. Lisäksi yhteys operoinnin ja ylläpidon välillä on yhä tärkeämpää, ja ALM-työkalujen tulisikin integroitua näiden organisaation osastojen käyttämien työkalujen kanssa. (Rossberg, Ehn & Olausson 2014, 28.)

Forresterin tutkimuksen mukaan ALM:in 2.0+-versioon kuuluu aiempien versioiden yhteydessä mainittujen jäljitettävyyden, automaation ja raportoinnin lisäksi yhteistyö ja työn suunnittelu (Rossberg, Ehn & Olausson 2014, 28). Työn suunnittelu tarkoittaa suunnittelutoimintoja, kuten tehtävien määrittelyn ja niiden ohjaamisen käytettävissä oleville resursseille. Yhteistyö-käsite puolestaan sisältää vaatimuksen, jonka mukaan ALM 2.0+ -työkalujen on tuettava organisaatioiden rajat ylittävää jaettua kehitysympäristöä. Kyseisten työkalujen on tuettava tehokkaasti jakamista, yhteistyötä ja vuorovaikutusta ryhmien jäsenten kesken, ja tämän kaiken tulisi tapahtua ilman monimutkaisuuden lisääntymistä työympäristössä. (Rossberg, Ehn & Olausson 2014, 29.)

3.8.5 Devops

DevOps tarjoaa mahdollisuuden ratkaista ALM:in ongelmia. Se on lähes sama asia kuin aiemmin mainittu yhdistetty ALM-näkymä, mutta DevOps poikkeaa yhdistetystä ALM-näkymästä oleellisesti siten, että se tuo kehitys- ja operointihenkilöstöt lähemmäksi toisiaan. Tällöin poistuu työn luovutusvaihe kehityshenkilöstöltä operointihenkilöstölle, ja liikearvoa tuotetaan jatkuvalla kehityksellä ja operoinnilla. (Rossberg, Ehn & Olausson 2014, 29.)

Devops ei ole oma metodinsa, vaan se käyttää ketteriä metodeja ja prosesseja kuten Scrumia ja Kanbania. Sen avainkäsitteet ovat jatkuva kehitys, jatkuva integrointi ja jatkuva operointi. Oleellista DevOpsissa on työskennellä pienten muutosten parissa suurien julkaisujen sijaan, poistaa manuaaliset vaiheet automatisoimalla prosesseja sekä pitää kehitys- ja testausympäristöt mahdollisimman lähellä tuotantoympäristöä (Rossberg, Ehn & Olausson 2014, 30.)

Devopsin tarkoitus on optimioida aika, joka kuluu ohjelmiston siirtyessä kehitysvaiheesta tuotantovaiheeseen. Näin pystytään vastaamaan nopeasti markkinoilla tapahtuviin muutoksiin ja sieltä tuleviin vaikutteisiin, eli täyttämään yksi yrityksen menestymisen perusedellytyksistä. (Rossberg, Ehn & Olausson 2014, 30.)

3.8.6 ALM ja PPM

ALM ja PPM voivat tukea toisiaan hyvin. ALM-repositorion tieto voi olla mainio lähde PPM-työkalulle, ja näin ollen päätöksiä voidaan perustella PPM-työkalua käyttäen. Tämä vaatii yhteen suuntaan toimivan yhteyden työkalujen välillä. Hyvä integraatio repositorioiden välillä antaa projektiportfoliopäättäjille pääsyn tarkkaan ja ajankohtaiseen tietoon, mikä helpottaa päätöksentekoa. (Rossberg, Ehn & Olausson 2014, 30-31.)

Gartnerin mukaan viisi toimivasta integraatiosta hyötyvää PPM-aktiviteettia ovat käynnissä olevien projektien ja niiden tilan katselmointi, ohjelmistoportfolion resursseihin kohdistaman vaikutuksen katselmointi, lisäresurssien myöntäminen, projektin uudelleenpriorisointi sekä investointien tehokkuuden tarkastelu. Muun muassa näihin aktiviteetteihin voi löytää tärkeää dataa ALM-repositoriosta. (Rossberg, Ehn & Olausson 2014, 31.)

4 Konfiguraationhallinta

4.1 Taustaa

Ensimmäinen konfiguraationhallintastandardi julkaistiin Yhdysvaltojen ilmavoimien tahdolta. Tarkoituksena oli vastata aseteollisuuden tuotteiden monimutkaistumisen tuomiin haasteisiin; tuotteiden kehityksen venyminen moniin vuosiin johti siihen,

ettei kehitys pysynyt enää yksinomaan yhden henkilön tai ryhmän käsissä. (Leon 2015, luku 1.)

Ohjelmistojen monimutkaistuessa samanlainen ongelma kohdattiin myös niiden kohdalla, joten lukuisat organisaatiot, kuten Yhdysvaltain puolustusministeriö, IEEE, ANSI ja ISO loivat oman standardinsa vastatakseen ohjelmistokehityksen konfiguraatiohallinnan puutteen asettamiin haasteisiin. Nykyisin konfiguraationhallintaa sovelletaan valtaosassa ohjelmistoyrityksiä, ja konfiguraatiohallintaa helpottavien työkalujen kirjo on laaja. (Leon 2015, luku 1.)

IEEE on määritellyt konfiguraatiohallinnan käytännöksi, jolla sovelletaan teknistä sekä hallinnollista ohjausta ja tarkastelua, jotta voidaan tunnistaa ja dokumentoida konfiguraatioalkion toiminnalliset ja fyysiset erityispiirteet, hallita näiden erityispiirteiden muutoksia, dokumentoida ja raportoida muutosten käsittelyä ja toteutustilaa sekä varmistaa määriteltyjen vaatimusten toteutuminen. (Leon 2015, luku 1.)

Ohjelmiston konfiguraatiohallinta on käytäntö, jolla sovelletaan teknistä ja hallinnollista ohjausta ja tarkastelua. Käytäntö tarkoittaa tässä yhteydessä sitä, että konfiguraatiohallinnan käyttö edellyttää tiettyjen sääntöjen seuraamista. Nämä säännöt määritellään konfiguraatiohallintasuunnitelmassa, ja niitä sovelletaan sekä tekniseen että hallinnolliseen viitekehykseen. Sääntöjen noudattaminen varmistetaan jatkuvalla valvonnalla, ja konfiguraationhallinnan soveltaminen vaatiikin jatkuvan valvonnan mahdollistavan organisaatiorakenteen. Valvontaa suorittavan ryhmän koko ja rakenne määrittyvät projektin koon ja monimutkaisuuden mukaan. (Leon 2015, luku 1.)

Konfiguraationhallintafunktion tulee tunnistaa konfiguraatioalkiot ja dokumentoida niiden toiminnalliset ja fyysiset ominaisuudet. IEEE:n mukaan konfiguraatioalkio on laitteiston, ohjelmiston tai näiden molempien muodostama kooste, joka on määrätty konfiguraationhallinnalle ja jota käsitellään yksittäisenä entiteettinä konfiguraationhallintaprosessissa. Kun konfiguraatioalkiot on tunnistettu ja niiden ominaisuudet dokumentoitu, tulee konfiguraatiohallintajärjestelmän ohjailla ominaisuuksiin tehtäviä muutoksia siten, ettei muutoksia pysty tekemään ilman tarvittavia valtuuksia. (Leon 2015, luku 1.)

Konfiguraatiohallintajärjestelmän tulisi lisäksi taltioida koko muutoksenhallintaprosessi ja välittää taltioidut tiedot, kuten muutospyynnöt, arvioinnit, vaikutusanalyysit

ja muutosten hyväksynnät, kaikille osallisille. Tämä edellyttää muutoksenhallintaprosessin dokumentointia. Yksittäisen muutospyynnön tilaa tulee seurata ja taltioida sen koko elinkaaren ajalta. (Leon 2015, luku 1.)

Hallintajärjestelmässä tulisi olla myös mekanismi, joka varmistaa, että kehitetty ja toimitettu järjestelmä on täysin yhtäläinen vaatimusmäärittelyssä ja muissa asiaankuuluvissa dokumenteissa kuvatun järjestelmän kanssa. Kyseinen mekanismi mahdollistaisi pyydetyn järjestelmän toimittamisen auditoinnin ja verifikoinnin avulla. (Leon 2015, luku 1.)

IIIE jakaa konfiguraatiohallinnan funktiot neljään aktiviteettiin; konfiguraation tunnistamiseen, konfiguraation kontrollointiin, tilan kirjanpitoon ja tarkasteluun sekä katselmointiin. Konfiguraation tunnistaminen koostuu konfiguraatioalkioiden valitsemisesta sekä niiden toiminnallisten ja fyysisten ominaisuuksien kirjaamisesta tekniseen dokumentaatioon. Konfiguraation kontrollointi puolestaan koostuu evaluoinnista, koordinoinnista, hyväksymisestä tai hylkäämisestä ja konfiguraatioalkioihin kohdistuvien muutosten toteutuksesta. Tilan kirjanpito koostuu konfiguraation hallintaan tarvittavien tietojen kirjaamisesta ja raportoinnista. Tarkastelu taas toteutetaan konfiguraatiohallintajärjestelmän oikeanlaisen toiminnan varmistamiseksi. Tarkastelulla myös varmistetaan, että konfiguraation toiminnallisten vaatimusten täyttyminen on todettu testaamalla, ja että konfiguraatio sisältää kaikki toimitettavat entiteetit. (Leon 2015, luku 1.)

Aiellon ja Sachin mukaan kuusi konfiguraationhallinnan toiminnallista aluetta ovat lähdekoodin hallinta, koontiversiotekniikka, ympäristökonfiguraatio, muutosten hallinta, julkaisutekniikka ja käyttöönotto. Nämä kuusi aluetta kattavat kaikki konfiguraatiohallinnan toiminnat, eli

- jokaisen konfiguraatioalkion tunnistaminen ja hallinta
- oikeiden konfiguraatioalkioiden versioiden valinta koontiversioon
- kehityksen hallinta ja ylläpito
- Testi-, integraatio- ja tuotantoympäristöjen tarkistus muutosten tekemiseen käytettävän virallisen mekanismin olemassaolon varmistamiseksi
- yksittäisen julkaisun konfiguraatioalkioiden paketointi ja tunnistaminen
- paketoitujen julkaisujen luovuttaminen tuotannolle

Konfiguraationhallinta on kirjanpidon lisäksi nimeämiskäytäntöjä, linjauksia ja tehtävien muutosten suhteen kaikki osapuolet huomioon ottavia käytäntöjä. Konfiguraationhallinta ei sovi projekteihin sellaisenaan, vaan se on räätälöitävä yksittäisten projektien erityispiirteiden mukaan. Näitä piirteitä ovat muun muassa projektin koko, sen epävarmuus, kehitysprosessi ja asiakkaan osallisuuden laajuus. Konfiguraationhallinta-aktiviteettien käyttöä ohjataan konfiguraationhallintasuunnitelman avulla. (Leon 2015, luku 1.)

4.2 Konfiguraatiohallinnan tärkeys

Hyviä syitä konfiguraationhallinnan käyttöönotolle ovat muun muassa

- Ohjelmistotuotteiden, projektien ja kehitystiimien luonne
- Ohjelmistojärjestelmien monimutkaistuminen
- Ohjelmistojen kysynnän kasvu
- Ohjelmistojen muuttuva luonne ja tarve muutoksenhallinnalle (Leon 2015, luku 4).

4.2.1 Ohjelmistotuotteiden, projektien ja kehitystiimien luonne

Ohjelmistotuotteet koostuvat monista eri komponenteista eri versioissa, ja niitä ajetaan monissa eri ohjelmisto- ja laitteistoalustoilla. Versiot ja muunnelmat ovat läheisesti toisiinsa liittyviä, ja ne muokkaantuvat, korruptoituvat ja tuhoutuvat ilman minkäänlaista ohjausmekanismia. (Leon 2015, luku 4.)

Ohjelmistokehitysprojektit ovat huomattavain vaikeita hallita, sillä niissä on todella paljon muuttuvia tekijöitä. Esimerkiksi vaatimuksissa, koodissa sekä kehitys- ja ajoympäristöissä tehtävät muutokset voivat johtaa kaaokseen, ja tämän estämiseksi projektien haltijoiden on käytettävä esimerkiksi seuraavia työkaluja (Leon 2015, luku 4):

- Työkalu, joka varmistaa muutosten tapahtumisen järjestelmällisesti ja tieteellisellä tavalla
- Työkalu, joka estää yhteydenpidon katkeamisen projektin osapuolien välillä
- Työkalu, joka varoittaa projektin haltijoita tulevista vikatilanteista
- Työkalu, joka pitää kirjaa kaikesta, jotta virheiden korjaus on helpompaa niiden ilmetessä

Projektien haltijat tarvitsevat lisäksi työkaluja, joiden avulla eri kulttuureista tulevat sovelluskehittäjät kykenevät kommunikoimaan keskenään. Työkalujen tulisi myös pitää tarkkaa kirjaa yksittäisen kehittäjän työskentelystä, jotta kehittäjän poistuessa

projektista hänet voidaan korvata uudella kehittäjällä mahdollisimman sujuvasti. (Leon 2015, luku 4.)

Ohjelmistotuotteiden, -projektien ja -tiimien erityislaatuisuus siis tekee ohjelmistojen kehityksestä uniikkia ja monimutkaista. Esimerkiksi konfiguraationhallintatyökalujen käyttö mahdollistaa turvallisen ohjelmistokehityksen, ja kokemus on osoittanut, että ero onnistuneen ja epäonnistuneen projektin erottaa toisistaan oikeanlaisten ohjausmekanismien ja tieteellisten mekanismien käyttö. (Leon 2015, luku 4.)

4.2.2 Ohjelmistojen monimutkaistuminen ja lisääntynyt kysyntä

Tietoteknisten ratkaisujen lisääntyminen arkipäiväisessä elämässä tarkoittaa myös ohjelmistotuotannon kasvua. Kysynnän kasvaessa kasvavat myös ohjelmistokehittäjille asetetut tuottavuusvaatimukset, mutta tuottavuuden kasvu jää reilusti vaatimusten kasvun jälkeen. Ohjelmistoja kehittäville tahoille onkin haasteellista tuottaa korkealaatuisia ohjelmistoja käytettävissä olevan ajan ja budjetin puitteissa. Monimutkaisuus puolestaan näkyy siinä, että ohjelmistojen koko on kasvanut, ja ohjelmiston tuottamisen kustannukset ovat kasvaneet kehityksen alkuhistorian tuhansista dollareista satoihin miljooniin dollareihin. Tämän lisäksi yksittäisten ohjelmistojen tuotanto on hajaantunut eri puolille maailmaa, kun ne ohjelmistotuotannon alkutaipaleella kehitettiin yleensä yhdessä sijainnissa. (Leon 2015, luku 4.)

Lisääntynyt ohjelmistojen kysyntä, tarve muokata ja ylläpitää olemassa olevia ohjelmistoja, ohjelmistokehityksen monimutkaistuminen sekä ohjelmistojen käyttö kriittisissä sovelluksissa ovat johtaneet siihen, ettei sovelluksia voi kehittää kuten ohjelmistotuotannon alkuaikoina. (Leon 2015, luku 4.)

4.2.3 Ohjelmistojen muuttuva luonne ja tarve muutoksenhallinnalle

Lehmanin jatkuvan muutoksen lain mukaan kaikki käytössä olevat suuret ohjelmistojärjestelmät muuttuvat jatkuvasti, sillä vasta järjestelmän käytön kautta selviää mahdolliset lisäominaisuudet. Kyseiset järjestelmät muuttuvat niin kauan, kun on taloudellisempaa tehdä muutoksia kuin luoda kokonaan uusi järjestelmä. Lehmanin mukaan myös vaatimukset täysin täyttävä järjestelmä muuttuu, sillä reaalimaailmassa ympäristö, jossa järjestelmää käytetään, on jatkuvan muutoksen alaisena. Näin ollen

järjestelmän on myös mukauduttava ympäristöönsä. Muutokset ovat siis varmoja ohjelmistojen ollessa kyseessä, ja niitä on kyettävä hallitsemaan esimerkiksi konfiguraationhallinnan avulla. (Leon 2015, luku 4.)

4.3 Konfiguraatiohallinnan edut

Hallitsematon ohjelmistokehitysprosessi johtaa helposti käytettävissä olevan ajan ja budjetin ylitykseen, sekä johtaa heikkolaatuisiin ohjelmistoihin. Konfiguraationhallinta on käytäntö, jonka avulla voi hallita muutoksia ja parantaa kommunikaatiota, ja jolla voidaan nostaa kehittäjien tuottavuutta vähentämällä epäjärjestystä, turhaa työtä ja ylimääräistä vaivannäköä. (Leon 2015, luku 4.)

Konfiguraationhallintajärjestelmää käyttämällä voidaan parantaa asiakaspalvelun laatua ja lisätä asiakastyytyväisyyttä, sillä järjestelmä helpottaa virheiden löytämistä ja lisäominaisuuksien sijoitusta, ja tällä tavoin nopeuttaa asiakkaiden kohtaamien ongelmien ratkaisua. (Leon 2015, luku 4.)

Hyvä konfiguraationhallintajärjestelmä automatisoi sovelluksien muutos- ja käyttöönottoprosessit. Se parantaa sovellusten laatua, pienentää virheiden määrää, lyhentää kehitys- ja muutossyklejä, pienentää markkinointiin tarvittua aikaa, automatisoi ja tehostaa koontiversioiden ja julkaisujen hallintaa sekä tehostaa koko organisaation työntekijöiden tuottavuutta. (Leon 2015, luku 4.)

Projektin tilan kirjanpito antaa projektin ja organisaation johdolle reaaliaikaista tietoa projektin tilanteesta. Nämä tiedot sisältävät mitattavia suureita, jotka kertovat muun muassa ohjelmistokehityksen laadusta ja prosessien kehityksestä. Kyseiset suureet antavat johdolle paremmat valmiudet ohjata projektia niin, että se pysyy aikataulussa. (Leon 2015, luku 4.)

Konfiguraatiohallintajärjestelmää käyttämällä voidaan pitää kehittäjät informoituna komponenttien päivityksistä, varmistaa uusimpien komponenttiversioiden saatavuus, estää yhtäaikaisista päivityksistä syntyvät ongelmat sekä varmistaa kommunikoinnin katkeamattomuus. Näin järjestelmällä saadaan säästettyä kehittäjien työaikaa, ja säästetty aika voidaan käyttää varsinaiseen kehitystyöhön. (Leon 2015, luku 4.)

Konfiguraationhallinta auttaa selkeyttämään monimutkaisia ohjelmistokokonaisuuksia projektin osakkaille. Se tunnistaa kaikki projektiin kuuluvat komponentit, tallentaa niiden fyysiset ja toiminnalliset ominaisuudet sekä luokittelee ja nimeää ne järjestelmällisesti ja tieteellisesti. Komponentit tallennetaan useaan eri paikkaan turvallisuussyistä. Konfiguraationhallintaa käytettäessä projektin haltija pysyy ajan tasalla projektin tilasta, ja sillä voidaan suorittaa tuotteelle tarkastuksia ja katselmointeja asiakkaiden asettamien vaatimusten täyttymisen varmistamiseksi. (Leon 2015, luku 4.)

Konfiguraatiohallinta estää luvattomien muutosten tekemisen projektin komponentteihin asettamalla katselmoidut ja hyväksytyt konfiguraatioalkiot perustietoineen hallittuihin kirjastoihin, joihin pääsy on rajoitettu. Hallinta pitää konfiguraatioalkioista muutoslokeja, joista voi nähdä kunkin alkion kehityskaaren. Nämä lokitiedot auttavat ongelmien selvittämisessä ja virheiden löytämisessä. Hallintajärjestelmissä on usein turvasäilöt, joissa hallittujen kirjastojen alkioiden viimeisimpiä versioita säilytetään. Järjestelmä pitää myös kirjaa yksittäisten työntekijöiden projektiin liittyvistä aktiviteeteista, mikä helpottaa vanhojen työntekijöiden korvaamista uusilla. (Leon 2015, luku 4.)

Konfiguraationhallintajärjestelmän käyttö helpottaa ohjelmien uudelleenkäyttöä. Se pitää kirjaa komponenttien historiasta, nykytilasta ja tulevaisuudesta, ja se määrittää komponenteille luokitukset ja nimet. Näin ollen komponenttien tunnistus ja haku helpottuvat, ja niitä voidaan helposti käyttää myöhemmissä versioissa tai muissa projekteissa. (Leon 2015, luku 4.)

Ohjelmistosta syntyvät kulut voidaan jakaa lyhyen ja pitkän aikavälin kuluihin. Lyhyen aikavälin kulut syntyvät esimerkiksi suunnittelusta, kehityksestä ja testauksesta. Pitkän aikavälin kulut tulevat ohjelmiston operoinnista ja ylläpidosta, ja ne muodostavat jopa 75% ohjelmiston elinkaaren aikana syntyneistä kustannuksista. Ohjelmiston ylläpito voidaan jakaa kolmeen alakategoriaan; korjaavaan ylläpitoon, mukauttavaan ylläpitoon ja täydentävään ylläpitoon. Korjaavassa ylläpidossa korjataan järjestelmään jääneitä virheitä, joita ei ole testauksessa havaittu. Mukauttavassa ylläpidossa taas ohjelmistoa muokataan, jotta se toimisi uudessa ajoympäristössä tai uusien rajapintojen kanssa. Täydentävässä ylläpidossa puolestaan lisätään järjestelmään uusia toimintoja ja ominaisuuksia. Ohjelmiston ylläpitovaihetta voi helpottaa systemaattisella

kehitystyöllä, täydellisellä dokumentoinnilla, siihen kuuluviin ohjelmiin tehtyjen muutoksien kirjaamisella ja ohjelmien keskinäisten riippuvuuksien määrittelyllä. Nämä asiat ovat saavutettavissa mekanismeilla, jotka varmistavat sekä suunnittelun että kehityksen systemaattisuuden ja ohjailtavuuden. Helpottunut ylläpitovaihe tarkoittaa ajansäästöä, mikä puolestaan pienentää ylläpitovaiheessa syntyviä kustannuksia. (Leon 2015, luku 4.)

Ohjelmistokehityksen hyvä laadunhallinta tarkoittaa sitä, että pyritään ongelmien korjaamisen lisäksi löytämään niiden juurisyyt. Nämä juurisyyt voidaan löytää kausaalisella analyysillä, mutta tällöin tarvitaan muodollinen mekanismi ongelmien raportointiin tai vikojen kirjanpitoon ja paikannukseen. (Leon 2015, luku 4.)

Kun kehitysprosessissa käytetään järjestelmää, joka osaa sijoittaa koodin oikein ja pitää kirjaa siitä, miksi, mihin ja milloin muutoksia on tehty, on ohjelmiston kehitys ja ylläpito helpompaa. Tällaisella järjestelmällä voidaan välttää esimerkiksi tilanne, jossa dokumentointi jää jälkeen, tai tilanne, jossa eri kehittäjät käyttävät komponenttien tai ohjelmien eri versioita. (Leon 2015, luku 4.)

Ohjelmistossa esiintyvien virheiden kirjaus, kategorisointi, syiden analysointi ja ratkaisujen kirjaaminen edesauttavat niiden korjaamista jatkossa, sillä ratkaisuista voi oppia jotain uusien samanlaisten virheiden ilmentyessä. (Leon 2015, luku 4.)

Ohjelmistot ovat nykyään niin suuria, että niiden kehittämiseen tarvitaan useampi kuin yksi henkilö. Mikäli kehitysympäristö on liian henkilökeskeinen, voi projekti olla uhattuna mikäli yksi tai useampi avainhenkilö poistuu siitä. Ohjelmistokehityksen tulisikin olla prosesseihin tukeutuvaa, ei henkilöihin. Tähän tarvitaan järjestelmällistä ja kurinalaista ympäristöä, joka tukee tiimityöskentelyä ja yhteistyötä. (Leon 2015, luku 4.)

Vaatimuksien sekä niihin tehtyjen muutoksien dokumentointi tarvitsee prosessin, jotta voidaan varmistaa, että sovellus on sellainen kuin asiakas on tilannut. Tarkasteluilla ja katselmoinneilla pystytään tarkistamaan vaatimusten täyttyminen kehitysprojektin joka vaiheessa. (Leon 2015, luku 4.)

Ohjelmistokehityksen muuttuneen luonteen takia tarvitaan järjestelmä, joka hallitsee ja ohjaa muutoksia sekä pitää huolen laadunvarmistuksesta. Pitkän aikavälin onnistumiseen tarvitaan prosesseihin nojaavaa järjestelmää. Konfiguraationhallintaa käyttämällä voi vastata näihin haasteisiin, ja se onkin lähes pakollinen nykypäivän ohjelmistokehitysympäristössä projektista riippumatta. Siihen liittyvien toimintojen tulisi olla valmiina mahdollisimman aikaisessa vaiheessa projektia. Konfiguraatiohallintaan liittyvät käytännöt tulisi tallettaa konfiguraatiohallintasuunnitelmaan niiden suunnittelun ja hyväksymisen jälkeen, ja tämän tulisi tapahtua vaatimusmäärittelyn hyväksymisen aikoihin. (Leon 2015, luku 4.)

4.4 Konfiguraatiohallinnan käsitteet

Konfiguraationhallinta on kokoelma aktiviteetteja, joita suoritetaan koko projektin elinkaaren ajan vaatimusanalyysistä ylläpitoon, ja se on tärkeää, sillä ohjelmistot ovat jatkuvan muutoksen alaisina. Hallitsemattomat muutokset johtavat ongelmiin, joten muutoksia on ohjailtava ja hallittava. (Leon 2015, luku 5.)

Ohjelmistokehitysprojektista syntyy kolme tuotetta; ohjelmat, dokumentaatio ja data. Nämä tuotteet yhdessä muodostavat ohjelmiston konfiguraation. IIIE määrittelee ohjelmistokonfiguraation fyysisinä ja toiminnallisina ominaisuuksina, jotka on määritelty teknisessä dokumentaatiossa tai saavutettu lopputuotteessa. Konfiguraationhallintajärjestelmä tunnistaa edellä mainitut tuotteet ja tallettaa niiden ominaisuudet ja suhteet. Tämä olisi helppo operaatio, mikäli ohjelmistot eivät olisi alttiina jatkuvalle muutokselle. (Leon 2015, luku 5.)

Bersoffin ensimmäisen järjestelmätekniikan lain mukaan järjestelmä muuttuu ja tarve muuttaa sitä on olemassa riippumatta siitä, missä kohdin ohjelmiston elinkaaressa ollaan. Konfiguraationhallinta auttaa toimimaan näiden muutosten kanssa, ja se myös varmistaa tarkasteluiden ja katselmointien avulla, että ohjelmisto toteuttaa sille asetetut vaatimukset. (Leon 2015, luku 5.)

Näin ollen konfiguraationhallinta voidaan määritellä joukoksi aktiviteetteja, joiden päätarkoitukset ovat konfiguraatioalkioiden tunnistus, alkioiden ominaisuuksien, piirteiden ja niiden keskinäisten riippuvuuksien kirjaaminen, alkioiden tarkkailu, alkioihin

tehtyjen muutosten hallinta, alkioiden muutosprosessien dokumentointi ja raportointi sekä alkioille asetettujen vaatimusten täyttymisen varmistaminen. (Leon 2015, luku 5.)

Konfiguraationhallintajärjestelmällä tarkoitetaan työkaluja, suunnitelmia ja toimenpiteitä, joilla konfiguraationhallintaa sovelletaan projekteissa. Se on konfiguraationhallintaa toteuttava kokoelma aktiviteetteja ja henkilöstöä sekä muita resursseja. (Leon 2015, luku 5.)

5 Versionhallintajärjestelmät

5.1 Yleistä

Versionhallinta on osa konfiguraatiohallintaa, ja sen tarkoitus on mahdollistaa ohjelmistokomponenttien ja niitä sisältävien järjestelmien, toisin sanoen codelinejen ja baselinejen eri versioiden seuraaminen. Codeline on lähdekoodiversioista muodostuva ketju, missä myöhemmät versiot on kehitetty aiemmista versioista, ja se koskee yleensä järjestelmän komponentteja. Baseline on järjestelmän määritelmä, eli se määrittelee muun muassa järjestelmään sisältyvät ohjelmistokomponenttien versiot, käytetyt kirjastot sekä asetustiedostot, ja eri baselineilla voi erotella esimerkiksi eri asiakkaille räätälöidyt järjestelmän versiot. (Sommerville 2016, 735.)

Versionhallintajärjestelmät erittelevät ja varastoivat komponenttiversioita sekä hallitsevat niiden saatavuutta. Järjestelmät voi jakaa ominaisuuksiensa perusteella kahteen eri ryhmään; keskitettyihin ja hajautettuihin versionhallintajärjestelmiin. Näillä ryhmillä on kuitenkin yhteisiä ominaisuuksia, joita ovat versioiden ja julkaisujen yksilöinti, muutoshistorian tallennus, tuki muista kehittäjistä riippumattomalle kehitykselle, tietovaraston hallinta sekä vaihtoehtoisesti myös tuki komponentteja keskenään jakaville projekteille. Lisäksi kaikkiin versionhallintajärjestelmiin liittyvät käsitteet projektirepositorio ja yksityinen työympäristö. Projektirepositorio pitää sisällään baselineissa käytettävät komponenttiversiot ja nämä versiot voidaan kopioida yksityisiin työympäristöihin, mistä niihin tehdyt muutokset voidaan päivittää takaisin projektirepositorioon. (Sommerville 2016, 735-736.)

5.2 Keskitetyt versionhallintajärjestelmät

Keskitetyissä versionhallintajärjestelmissä käytetään keskusrepositoriota, mihin kaikki muutoshistoria tallentuu. Kehittäjät lataavat ne komponentit, joita haluavat muokata, keskusrepositoriosta, tekevät komponentteihin muutokset ja palauttavat muokatut komponentit keskusrepositorioon, jolloin komponentista tallentuu uusi versio. Komponenttien muokkaus useamman useamman käyttäjän kesken tapahtuu niin, että kunkin käyttäjän palauttamasta muokatusta komponentista syntyy aina uusi, yksilöity versio keskusrepositorioon, ja käyttäjä saa versionhallintajärjestelmältä ilmoituksen, mikäli myös joku muu on muokannut ja palauttanut käyttäjän muokkaaman komponentin. (Sommerville 2016, 737.)

5.3 Hajautetut versionhallintajärjestelmät

Hajautetuissa versionhallintajärjestelmissä kehittäjät luovat työympäristöihinsä kloonit päärepositoriosta. Tässä lähestymistavassa on se etu, että käyttäjä voi päivittää muutoksia paikallisesti ilman internet-yhteyttä ja käyttäjä kykenee testaamaan järjestelmää työkoneellaan. Lisäksi kloonaus mahdollistaa sen, että päärepositorion tuhoutuessa se pystytään palauttamaan kokonaisuudessaan kloonista. Komponentteihin tehdyt muutokset tallennetaan paikalliseen repositorioon, mistä kehittäjä voi halutessaan lähettää muokatut komponentit päärepositorioon tai integraatiopäällikölle, kenellä on valta hyväksyä tehdyt muutokset päärepositorioon. Hajautetuissa versionhallintajärjestelmissä päärepositorio ei ole pakollinen, mutta ilman sitä voidaan päätyä tilanteeseen jossa kehittäjät kehittävät komponenttia käyttäen vanhoja versioita muista siihen liittyvistä komponenteista. Tämä voi johtaa tilanteeseen missä eheää järjestelmää ei saada luotua erillisistä kloonirepositorioista. Hajautetuissa järjestelmissä olevan edistyksellisen itsenäisen kehityksen tuen johdosta komponenteista on mahdollista kehittää itsenäisiä haarautumia, joissa komponentin kehitys voi olla hyvinkin erilaista. Nämä haaraumat voi myöhemmin yhdistää päärepositorioon yhdeksi versioksi, ja yhdistäminen toimii automaattisesti, mikäli haaroissa koodiin tehdyt muutokset eivät ole päällekkäisiä, eli muutoksia ei ole tehty esimerkiksi samaan funktioon. (Sommerville 2016, 737-739.)

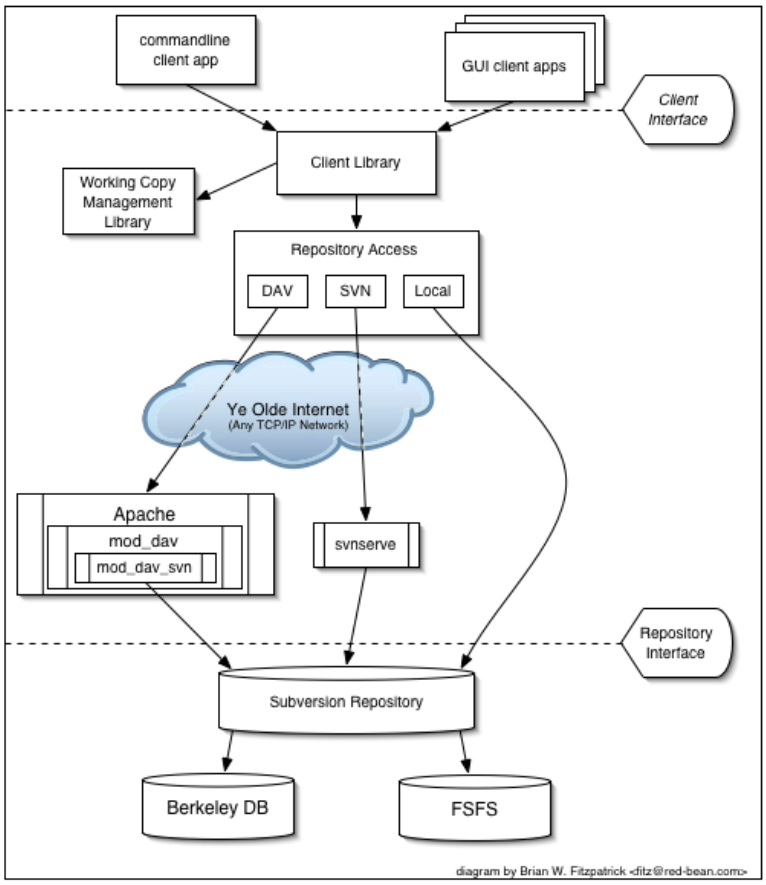
5.4 Subversion

5.4.1 Historiaa

Subversion on ilmainen, avoimen lähdekoodin keskitetty versionhallintajärjestelmä. Sitä alettiin kehittämään vuonna 2000 CollabNet Inc:n toimesta korvaamaan rajoittunut ja paljon virheitä sisältänyt CVS-versionhallintajärjestelmä. Alkuperäiseen kehitystiimiin kuului joukko CVS-järjestelmään kyllästyneitä kehittäjiä, jotka pyrkivät luomaan uuden, CVS:iin perustuvan järjestelmän ilman sen vikoja. Subversion olikin johtava versionhallintajärjestelmä aikana, jolloin kovalevytila oli kallista ja internet-yhteydet hitaita. (Collins-Sussman, Fitzpatrick & Pilato 2011, xiv-xv.)

5.4.2 Rakenne

Kuviossa 1 näkyy alemman poikittaisen katkoviivan alapuolella Subversion-repositorio, jossa säilytetään kaikkea versioitua dataa. Ylemmän poikittaisen katkoviivan yläpuolella puolestaan on Subversion-asiakasohjelma, joka hallitsee paikallisia osia repositorion versioidusta datasta. Katkoviivojen välissä taas on repositorioyhteyskerros, jossa yhteys asiakasohjelman ja repositorion välille muodostetaan suoraan tai tietoverkkojen ja palvelimien kautta. (Collins-Sussman, Fitzpatrick & Pilato 2011, xvi-xvii.)



Kuvio 1. Subversion-järjestelmän rakenne (Collins-Sussman, Fitzpatrick & Pilato 2011, xvi)

5.4.3 Keskeiset käsitteet

Joissain versionhallintajärjestelmissä tiedostot niin sanotusti lukitaan jonkun kehittäjän muokatessa niitä niin, etteivät muut kehittäjät pysty tekemään niihin muutoksia, mutta Subversion käyttää kopioi-muokkaa-yhdistä -mallia komponenttien päällekkäisen kehityksen hallintaan. Tässä mallissa kehittäjät muokkaavat kopioimaansa samaa koodia samanaikaisesti, minkä jälkeen he palauttavat koodin projektirepositorioon, missä ne yhdistetään yhdeksi uudeksi versioksi. Yhdistämisen onnistumisen tarkistaminen hoituu manuaalisesti, mutta Subversionissa on yhdistämistä helpottavia ominaisuuksia. (Collins-Sussman, Fitzpatrick & Pilato 2011, 4-5.)

Subversion-repositorio on abstrakti entiteetti, jota kontrolloidaan sen omien kirjastojen ja työkalujen avulla. Kun kehittäjä palauttaa repositorioon muokkaamiaan tiedostoja asiakasohjelman kautta, niin repositorio joko hyväksyy tai hylkää kaikki muutok-

set. Aina kun muutokset hyväksytään, luo repositorio uuden, järjestelmällisesti numeroidun tilan sen sisältämälle tiedostopuulle, eli tavallaan ottaa valokuvan koko tietojärjestelmästä tehtyjen muutoksien jälkeen. Tätä tilaa kutsutaan revisioksi. (Collins-Sussman, Fitzpatrick & Pilato 2011, 7.)

Työkopio on käyttäjän työympäristössä sijaitseva kopio jostain Subversion-repositorion revisiosta. Työkopio on täysin yksityinen, eikä se ole päärepositorio tee siihen mitään muutoksia, ellei kehittäjä erikseen niin tahdo. Kun halutut muutokset on tehty, voi kehittäjä julkaista ne päärepositorioon. Mikäli joku muu on tehnyt muutoksia käyttäjän työkopiota vastaavaan revisioon, voi käyttäjä ladata nuo muutokset yhdistettäväksi työkopionsa kanssa. Nämä muutokset tarkistetaan työkopion juuressa sijaitsevassa .svn-hakemistossa, eli hallintahakemistossa sijaitsevien tiedostojen avulla. Kyseiset tiedostot sisältävät tiedot jokaisesta työkopion tiedostosta, tärkeimpinä näistä revisio johon tiedosto perustuu sekä aikaleima viimeisestä kerrasta jona tiedosto on päivitetty päärepositoriosta. Subversion voi näillä tiedoilla määrittää kullekin työkopion tiedostolle tilan, ja nämä neljä tilaa ovat seuraavat: ”muuttumaton ja ajankohtainen”, ”muutettu ja ajankohtainen”, ”muuttumaton ja vanhentunut” sekä ”muutettu ja vanhentunut”. ”Muuttunut” ja ”muuttumaton” viittaavat tiedoston työkopion tilaan, kun taas ”ajankohtainen” ja ”vanhentunut” määrittelevät sen muokkaamattoman tilan suhteen päärepositoriossa olevaan revisioniin. (Collins-Sussman, Fitzpatrick & Pilato 2011, 9-10.)

5.4.4 Perustoiminnot

Subersion-repositorio sisältää tyypillisesti useiden eri projektien koodia ja tiedostoja. Kehittäjä voi tällöin halutessaan kopioida työkopioonsa repositoriossa sijaitsevan, yksittäistä projektia vastaavan alipuun. Kyseinen alipuu sisältää yleensä Subversion-projektissa määritellyn tiedostorakenteen, jossa ylimpänä ovat trunk-, branches- ja tags-nimiset hakemistot. Trunk-hakemistossa sijaitsee kehityksen päälinja, branches-hakemistossa päälinjasta tai muista kehityshaaroista luodut kehityshaarat ja tags-hakemisto, joka sisältää eri kehityshaaroista luotuja tilannevedoksia. (Collins-Sussman, Fitzpatrick & Pilato 2011, 17.)

Tyypillinen työsykli alkaa työkopion päivittämisellä. Päivittämisen jälkeen käyttäjä tekee haluamansa muutokset; päivitykset, tiedostojen luonnit ja poistot sekä tiedostojen ja hakemistojen kopioinnit ja siirrot. Tämän jälkeen käyttäjä voi tarkastella tekemiään muutoksia ja tehdä korjauksia tarvittaessa. Ennen viimeistä vaihetta, eli muutosten päivittämistä repositorioon, tulee käyttäjän vielä päivittää työkopionsa ja ratkaista mahdolliset muiden käyttäjien muutoksista aiheutuvat konfliktit. (Collins-Sussman, Fitzpatrick & Pilato 2011, 19-20.)

Subversion-repositorio pitää kirjaa kaikista siihen päivitetyistä muutoksista. Käyttäjä voi tarkastella paikallisesti tehtyjä muutoksia, verrata työkopiota repositorioon sekä vertailla kahta revisiota keskenään. (Collins-Sussman, Fitzpatrick & Pilato 2011, 33-35). Hän voi myös tarkastella jonkin revision yksittäisen tiedoston sisältöä ilman työkopion tiedostoston päivittämistä. (Collins-Sussman, Fitzpatrick & Pilato 2011, 37). Päivitetyt muutokset tallentuvat repositorion lokiin, josta käyttäjä voi halutessaan tarkastella revisioissa tehtyjä muutoksia, muutosten ajankohtia ja mahdollisia päivityksen yhteydessä tehtyjä lokiviestejä (Collins-Sussman, Fitzpatrick & Pilato 2011, 35). Käyttäjä voi myös halutessaan korvata työkopionsa haluamallaan vanhalla revisiolla (Collins-Sussman, Fitzpatrick & Pilato 2011, 39-40).

5.4.5 Lisätoiminnot

Subversionin revisioiden hallintaan voi käyttää valmiita avainsanoja. Nämä avainsanat ovat HEAD, BASE, COMMITED ja PREV. Head vastaa repositorion viimeisintä revisiota, BASE työkopion muokkaamattoman tiedoston revisiota, COMMITED viimeisintä revisiota missä tiedostoa on muokattu sekä PREV-revisio COMMITED-revisiota edeltävää revisiota. (Collins-Sussman, Fitzpatrick & Pilato 2011, 47-48.)

Subversion mahdollistaa peg revision -algoritmin käytön. Kyseisen algoritmin avulla voidaan eritellä samoissa poluissa sijanneiden samannimisten tiedostojen historia. Kun esimerkiksi luodaan tiedosto nimeltä x, päivitetään se repositorioon, poistetaan se repositoriosta muutaman päivityksen jälkeen, luodaan se uudelleen sekä päivitetään se repositorioon, on molemmilla repositoriossa olleilla x-nimisillä tiedostoilla oma historiansa. Oletuksena Subversion käsittelee viimeksi viimeiseksi polkuun luotua tiedostoa, mutta peg-algoritmin avulla voidaan käsitellä myös ensiksi luodun tiedoston historiaa. (Collins-Sussman, Fitzpatrick & Pilato 2011, 49-50.)

Subversion tukee erilaisten ominaisuuksien määrittämistä versioiduille tiedostoille. Ominaisuuksilla tarkoitetaan tässä yhteydessä tiedostoihin ja hakemistoihin liitettävää, tekstimuotoista metadataa. Ominaisuudet, kuten repositorion tiedostot ja hakemistotkin, ovat versioituja, ja ne voivat sisältää mitä informaatiota tahansa. Ne voivat olla Subversionin omia tai kehittäjän luomia. (Collins-Sussman, Fitzpatrick & Pilato 2011, 53.) Subverisonin omia ominaisuuksia ovat esimerkiksi MIME-tyyppi, eli tiedostotyyppi, sekä End-Of-Line-merkki, eli rivinvaihtomerkki (Collins-Sussman, Fitzpatrick & Pilato 2011, 61-63). Lisäksi on avainsanaominaisuus, joka muokkaa tiedostoa repositorioon päivittämisen yhteydessä niin, että tiedostoon lisätään halutun avainsanan yhteyteen haluttua tietoa, esimerkiksi päivämäärä tai revisionumero. Kehittäjä taas voi esimerkiksi lisätä tiedostoon haluamansa lisenssin tai yksinkertaisen kommentin tiedoston sisällöstä. (Collins-Sussman, Fitzpatrick & Pilato 2011, 54-58.) Tiedostojen ja hakemistojen lisäksi ominaisuuksia on mahdollista määrittää myös revisioille, mutta rajoitteena on tällöin se, ettei ominaisuuksista kerry muutoshistoriaa repositorioon (Collins-Sussman, Fitzpatrick & Pilato 2011, 53).

Subversionissa on mahdollista määrittää järjestelmän sivuuttamat tiedostot, eli työhakemiston tiedostot joita ei päivitetä repositorioon. Tämä on mahdollista muun muassa global-ignores –asetuksella, jossa määritellään rakenteet, joiden perusteella tiettyjä merkkijonoja sisältävät tiedosto- ja hakemistonimet sivuutetaan päivityksen yhteydessä. Hakemistokohtaiset sivuutukset on mahdollista määritellä svn:ignore-ominaisuudella. (Collins-Sussman, Fitzpatrick & Pilato 2011, 63-66.)

Ominaisuuksien lisäksi tiedostoille on mahdollista määrittää avainsanoja, jotka päivittyvät aina, kun tiedostot päivitetään päärepositorioon. Subversion määrittelemään avainsanalistaan kuuluvat sanat Date, Revision, Author, HeadURL, Id ja Header. Date-avainsana määrittelee viimeisimmän muokkausajankohdan, Revision-avainsana viimeisen revisionin jossa tiedostoa on muokattu, Author-avainsana viimeisimmän muokkaajan ja HeadURL tiedoston viimeisimmän version polun repositoriossa. Header-avainsana puolestaan on tiivistelmä kaikista edellä mainituista avainsanoista, ja Id-avainsana on tiivistelmä ilman tiedoston viimeisimmän version repositoriopolkua. (Collins-Sussman, Fitzpatrick & Pilato 2011, 67-68.)

Subversionissa voi kopioida hakemistoja repositoriosta ei-rekursiivisesti, eli esimerkiksi lataamalla hakemiston niin, että sen tiedostot ladataan mutta alihakemistoja ei

ladata. Rekursiivisuuden taso voidaan määritellä depth-asetuksella, kun repositorio ladataan työkopioon. (Collins-Sussman, Fitzpatrick & Pilato 2011, 71-72.)

Lukot ovat Subversionissa varsin hyödyllinen ominaisuus. Kehittäjä voi rajoittaa yksittäisen, repositoriossa olevan tiedoston muokkauksen itselleen niin, ettei kukaan muu voi päivittää kyseistä tiedostoa repositorioon. Sen lisäksi, että lukkoja voidaan luoda ja poistaa kehittäjän toimesta, voivat muut kehittäjät murtaa lukon halutessaan. Tällöin heidän on kuitenkin pakko olla tietoisia lukon olemassaolosta. (Collins-Sussman, Fitzpatrick & Pilato 2011, 75-76.)

Subversionissa on mahdollista määrittää alihakemistolle erillinen lähde, esimerkiksi eri revisiossa sijaitseva tai jopa eri repositoriossa sijaitseva alihakemisto. Tämä määritelmä, svn:externals, lukeutuu myös aiemmin mainittuihin, Subversionin valmiisiin ominaisuuksiin. (Collins-Sussman, Fitzpatrick & Pilato 2011, 82.)

Muutoslistalla on mahdollista yhdistää tiedostoja ja hakemistoja yhteisen nimen alle niin, että ne voidaan tämän yhteisen nimen avulla päivittää repositorioon. Muutoslistan nimen avulla voidaan suorittaa repositorioon päivittämisen lisäksi myös esimerkiksi muutoksien vertailua. (Collins-Sussman, Fitzpatrick & Pilato 2011, 87-88.)

5.4.6 Haaraumat ja niiden yhdistäminen

Kehittäjä voi halutessaan luoda päälinjasta haarauman ja kehittää sitä haluamallaan tavalla. Näin voidaan esimerkiksi välttää bugeja päälinjassa ja antaa ”työrauha” muille kehittäjille niin, etteivät heidän viimeisimpään päälinjan revisioon tekemänsä muutokset ole vaarassa. Luodut haarautumat ovat ”halpoja” sillä ne eivät luo tiedostoja kokonaan uudelleen, vaan linkittävät lähdehaaran luotuun haaraan niin, että uuteen haaraan tallentuvat varsinaisesti ainoastaan kehittäjän myöhemmin lisäämät tiedostot sekä hänen päivittämänsä muokkaukset. Vaikka haarat syntyvät linkittämisen avulla, näyttäytyvät ne kuitenkin repositoriossa siten kuin ne sisältäisivät alkuperäisestä haarasta tai päähaarasta linkitetyt tiedostot. (Collins-Sussman, Fitzpatrick & Pilato 2011, 96-99.)

Kun kehittäjä on tehnyt haluamansa muokkaukset luomaansa haaraan, voi hän yhdistää haaran jonkin toisen haaran tai projektin päälinjan kanssa. Yhdistämisestä tallentuu oma ominaisuutensa, svn:mergeinfo-ominaisuus, yhdistämisen kohteena oleviin

tiedostoihin ja hakemistoihin. Kyseinen ominaisuus ilmaisee, mitkä kunkin lähdehaaran muutokset on ladattu kohteeseen. Haaraan voi myöhemmin myös päivittää sen vanhempaan haarautumiseen tehdyt muutokset, joko kaikki muutokset tai sitten kehittäjän valitsemat. Yhdistämiskomentoa voi myös käyttää haaraan tehtyjen muutoksien kumoamiseen. (Collins-Sussman, Fitzpatrick & Pilato 2011, 102-106.)

Subversionissa kehittäjä voi luoda halutessaan tagin haluamansa haaran tai päähaaran tilasta. Tagi on oikeastaan halutun haaran tilasta otettu snapshot, jonka kehittäjävoi nimetä haluamallaan tavalla. Tagin voi luoda repositorioon myös kehittäjän työhakemiston tilasta. (Collins-Sussman, Fitzpatrick & Pilato 2011, 130-131.)

5.5 Git

Git on hajautettu versionhallintajärjestelmä, eli kehittäjä ei sitä käyttäessään luo vain työkopiota työasemalleen, vaan kokonaisen repositorion kaikkine historiatietoineen ja haaroineen. Se syntyi Linux-kehittäjäyhteisön toimesta vuonna 2005, kun yhteisön aiemmin käyttämä BitKeeper-versionhallintajärjestelmä muuttui maksulliseksi Bitkeeper-yhtiön ja Linux-yhteisön yhteistyön loppuessa. Git kehitettiin yhteisön BitKeeper-järjestelmän parissa saatujen kokemusten pohjalta. (Chacon & Straub 2018, 13.)

Git luo tarkistussummat päivityksen yhteydessä päivitettäville hakemistoille ja tiedostoille. Näin voidaan varmistaa tiedon eheys. (Chacon & Straub 2018, 15.)

Tiedoston poistaminen Git-repositoriosta päivityksen yhteydessä ei poista sitä repositorion historiasta, joten kehittäjä voi myöhemmin halutessaan palauttaa sen (Chacon & Straub 2018, 15).

Git käsittelee tietoja niin, että se tallentaa aina päivityksen yhteydessä tilannevedoksen haaran tietojärjestelmän tilasta sekä viitteen sille. Muuttamattomia tiedostoja ei tallenneta uudelleen, vaan Git lisää päivitykseen viitteet edellisen päivityksen tiedostoihin. (Chacon & Straub 2018, 14.)

Kehittäjä voi joko kopioida olemassa olevan Git-repositorion tai luoda itse repositorion työkoneelleen (Chacon & Straub 2018, 24). Git-repositorion seuratuilla, eli repositorioon lisätyillä tiedostoilla on kolme tilaa; päivitetty (committed), muokattu (mo-

dified) sekä valmisteltu (staged). Päivitetty tila tarkoittaa sitä, että tiedosto vastaa viimeksi repositorioon päivitettyä tiedostoa. Muokattu tila taas tarkoittaa sitä, että tiedostoa on muokattu sen jälkeen, kun se viimeksi päivitettiin repositorioon. Valmisteltu tila puolestaan viittaa siihen, että tehdyt muutokset päivitetään repositorioon seuraavan päivityksen yhteydessä. (Chacon & Straub 2018, 15-17.) Työhakemistossa olevilla tiedostoilla puolestaan on kaksi tilaa; seurattu ja seuraamaton. Seuratut tiedostot ovat tiedostoja, jotka olivat viimeisessä tilannevedoksessa, ja ne voivat olla muokkaamattomia, muokattuja tai valmisteltuja. Seuraamattomat tiedostot puolestaan ovat tiedostoja, joiden olemassa oloa Git ei huomioi. (Chacon & Straub 2018, 26.)

Git status –komento näyttää työhakemistossa olevien tiedostojen tilan. Se listaa valmistellut sekä valmistelemattomat uudet tai muokatut tiedostot. (Chacon & Straub 2018, 26-29.) Kehittäjä voi lisätä uusia tai muokattuja tiedostoja valmisteluvaiheeseen git add –komennolla (Chacon & Straub 2018, 27-28). Tiedostot, joita ei haluta listattavan, tulee määritellä .gitignore-tiedostossa. Kyseisessä tiedostossa voidaan merkkijonorakenteet, joita vastaavat tiedostonimet jätetään huomiotta git status –komennon suoritusvaiheessa. (Chacon & Straub 2018, 30.) Git diff –komento kertoo muutoksista, joita ei ole vielä valmisteltu, ja staged-parametrilla se näyttää seuraavaan päivitykseen tulevat valmistellut muutokset (Chacon & Straub 2018, 32). Lopuksi kehittäjä voi päivittää valmistelemansa muutokset repositorioon git commit –komennolla. Päivityksen yhteydessä luotavaan tilannevedokseen lisätään automaattisesti luotu tai käyttäjän määrittelemä päivitysviesti. (Chacon & Straub 2018, 34-35.) Git log –komennolla kehittäjä voi tarkastella repositorioon tehtyjä päivityksiä. Komento näyttää päivitysten tarkastussummat, päivitysten tekijät, päivitysajankohdat ja päivitysviestit. (Chacon & Straub 2018, 38-39.) Tiedoston poistamiseksi repositoriosta käytetään git rm –komentoa, joka määrittelee tiedoston poistettavaksi seuraavassa päivityksessä (Chacon & Straub 2018, 36-37). Git mv –komento puolestaan määrittää tiedoston siirrettäväksi esimerkiksi toisen nimiseen tiedostoon seuraavassa päivityksessä (Chacon & Straub 2018, 38).

Repositorioon voi lisätä sekä poistaa etärepositorioita, minne kehittäjä voi päivittää tekemiään muutoksia, sekä ladata niiden kautta esimerkiksi jonkun toisen kehittäjän

tekemiä muutoksia omaan repositorioonsa. Kullekin paikalliselle repositorion haaralle voi myös määrittää omat haaransa etähaaran päivittämiseen ja siihen tehtyjen muutosten lataamiseen. (Chacon & Straub 2018, 48-51.)

Kehittäjä voi lisätä tekemäänsä päivitykseen tagin, minkä avulla päivityksen voi yksilöidä helposti. Tagi voi olla esimerkiksi versionumero. Tageihin voi lisätä haluamansa viestin, mutta kehittäjä voi luoda niin sanotun kevyen tagin, joka sisältää vain päivityksen tarkistussumman ja tagin nimen. Tagit eivät siirry automaattisesti etärepositorioon sinne päivittämisen yhteydessä, vaan käyttäjän pitää erikseen määrittää niiden siirtäminen. (Chacon & Straub 2018, 53-58.)

Kehittäjä voi halutessaan määrittää nimimerkkejä repositiossa käyttämilleen komennoille. Nimimerkeillä on mahdollista nopeuttaa Git-operaatioiden kirjoittamista. (Chacon & Straub 2018, 58.)

Git luo päivityksen yhteydessä päivitysobjektin, joka sisältää osoittimen kehittäjän lisäämään sisältöön. Objektissa on myös päivittäjän nimi sekä sähköpostiosoite, päivityksen yhteydessä lisätty viesti sekä osoittimet päivitystä välittömästi edeltäviin päivityksiin. Näitä osoittimia ei ole yhtään, mikäli kyseessä on ensimmäinen päivitys. Objektissa on yksi osoitin, mikäli sitä edeltävä päivitys on olemassa, ja osoittimia voi olla useampikin, mikäli päivitystä on edeltänyt haarojen yhdistäminen. (Chacon & Straub 2018, 60.)

Päivityksen yhteydessä Git luo tarkistussummat alihakemistoista, ja muodostaa repositorion juurihakemistosta alkavan, tarkistussummilla yhdistetyn tiedostopuun. Tämän tiedostopuun juuren osoitin lisätään päivitysobjektiin päivityksessä luodun viestin sekä päivitystä välittömästi edeltävän päivityksen osoittimen kanssa. Uuden haaran luonti Git-järjestelmässä toimii yksinkertaisuudessaan niin, että lähdehaarasta otetaan haluttuun päivitykseen viittaava osoitin ja liitetään se uuteen haaraan. Käytännössä luodaan uutta haaraa edustava tekstitiedosto, johon kyseinen osoitin sijoitetaan. Tämän jälkeen uuteen ja vanhaan haaraan tehdyt päivitykset ovat toisistaan riippumattomia. Kun uudessa haarassa on tehty halutut muutokset, voidaan se yhdistää takaisin lähdehaaraan. Mikäli lähdehaarassa ei ole tehty muutoksia, siirtyy sen osoitin kohdehaaran viimeiseen päivitykseen. Jos taas muutoksia on tehty, luo Git uuden päivityksen johon merkitään osoittimilla vanhemmiksi molempien haarojen

viimeisimmät päivitykset. Mikäli näissä molemmissa päivityksissä on muokattu saman tiedoston samaa osaa, ilmoittaa Git syntyneestä ristiriidasta, minkä jälkeen käyttäjä voi muokata yhdistetyn haaran tiedostoa, johon on Gitin toimesta merkitty ristiriidan aiheuttaneet kohdat, ja valita kohdista toisen tai luoda kokonaan uuden kohdan. Gitin ristiriitatilanteessa automaattisesti lisäämät rivit tulee myös poistaa. Muokkauksen jälkeen käyttäjä voi merkata tiedoston valmistelluksi ja päivittää uuteen, yhdistettyyn haaraan. (Chacon & Straub 2018, 60-75.)

Etäviitteet voivat olla osoittimia etäisiin repositorioihin, haaroihin tai tageihin. Etähaarojen tarkoitus on osoittaa etähaaran tila, eli viimeisin siihen tehty päivitys, viimeisenä kertana kun käyttäjä oli sen kanssa vuorovaikutuksessa. (Chacon & Straub 2018, 80-81.) Kehittäjä voi luoda paikalliseen repositorioonsa jäljityshaaroja, joiden tarkoitus on helpottaa ajan tasalla pysymistä etähaarojen kehityksen suhteen (Chacon & Straub 2018, 87-88).

Git tarjoaa vaihtoehdon haarojen yhdistämiselle. Tämä vaihtoehto on uudelleenperustaminen, eli rebase. Kyseinen ominaisuus mahdollistaa lähdehaarasta tehdyn haaran muutoksien yhdistämisen lähdehaaran viimeiseen päivitykseen niin, että haara pysyy erillisenä lähdehaarasta. Tällöin siis haarautumiskohta siirtyy päähaaralla viimeisimpään päivitykseen. (Chacon & Straub 2018, 90-99.)

Gitissä voi myös tallentaa tehtyjä muutoksia niin sanotusti jemmaan, stasheihin, ilman repositorioon päivittämistä. Näin käyttäjä voi palauttaa esimerkiksi tekemänsä muutokset muiden kehittäjien näkemättä niitä. (Chacon & Straub 2018, 226.)

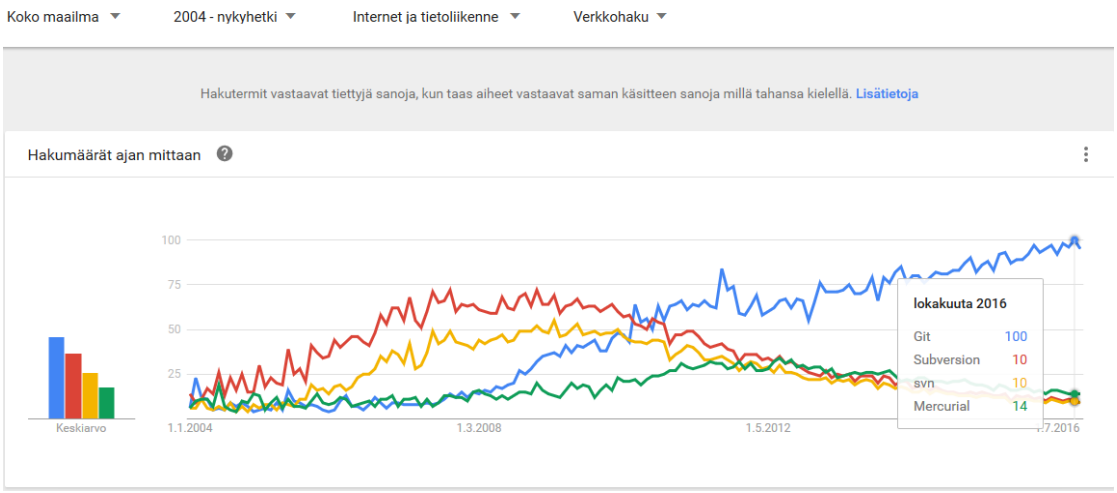
HEAD on osoitin nykyiseen haaraviittaukseen joka taas on osoitin viimeisimpään haarassa tehtyyn päivitykseen. HEAD-osoitinta ja haaraviittausta voi siirtää haluttuun päivitykseen niin, että repositorio käsittelee kyseistä päivitystä viimeisimpänä tehtynä päivityksenä. (Chacon & Straub 2018, 62.)

Gitissä kehittäjä voi luoda repositorioon alimoduuleja, joska ovat repositoriosta erillisiä alirepositorioita. Repositorio ja sen alimoduulit ovat toisistaan riippumattomia siinä mielessä, että Git seuraa niihin tehtyjä muutoksia erikseen, eli esimerkiksi alimoduuliin tehdyt muutokset eivät rekisteröidy repositorioon. (Chacon & Straub 2018, 296.)

5.6 Git vs. Subversion

Kuviossa 2 vertaillaan termien git, subversion, svn ja mercurial suhteellista suosiota verrattuna taulukon maksimisuosioon, jota vastaa luku 100. Kuviossa esiintyvä hakusana svn on Subversionin komentoriviltä ajettava asiakasohjelma (Collins-Sussman, Fitzpatrick & Pilato 2011, xvii). Mercurial taas on eräs hajautettu versionhallintajärjestelmä. Kuvion taulukon oikeanpuolimmaisin piste vaaka-akselilla vastaa hakusanojen suhteellisia määriä marraskuussa 2016, ja luvuista on selkeästi nähtävillä, että Git oli vertailluista versionhallintajärjestelmistä haetuin kyseisen kuukauden aikana. Lisäksi taulukosta näkee, että Git-hakusanan suosio on kasvanut viime vuosina samalla, kun muiden hakusanojen suosio on vähentynyt. Hakujen suhteellisesti korkein määrä ei luonnollisestikaan kerro siitä, että Git olisi paras vaihtoehto, mutta on todennäköistä että hakujen suhteellisesti korkein määrä tällä hetkellä korreloi kyseisen järjestelmän suosion kanssa.

Subversionin sivuille on listattu sitä kehityksessään käyttäviä avoimen lähdekoodin projekteja sekä niiden repositorioita (Subversion Testimonials n.d.). Seitsemän seitsämästätoista projektista oli siirtynyt käyttämään Git-järjestelmää, kun taas kahdeksan käytti Subversion-järjestelmää yhä. Jäljelle jääneistä projekteista Conectiva-projektia ei enää ole ja GNU Enterprise -projekti on siirtynyt käyttämään Bazaar-järjestelmää.



Kuvio 2. Google Trends -vertailu hakusanojen kesken

6 Vaihtoehdot Trac-järjestelmälle

6.1 Github

6.1.1 Lyhyesti

GitHub on koodinhostausalusta versionhallintaan ja yhteistyöhön. Se on monilla tavoin samanlainen kuin Trac, mutta se käyttää versionhallintaan Git-järjestelmää. Se myös eroaa Trac-järjestelmästä niin, että projektien wikisivut ovat omissa Git-repositorioissaan, ja projektien ylläpito on mahdollista pilvessä.

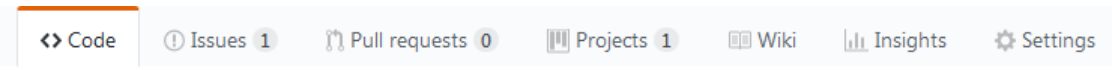
6.1.2 Projektisivu

Kuvio 3 näyttää esimerkkiprojektin etusivulla olevat välilehdet, joiden kautta käyttäjä voi hallita projektia ja tarkastella sen koodia ja dokumentaatiota.

Code-välilehdellä käyttäjä voi tarkastella haaroihin tehtyjä päivityksiä sekä haaroissa olevia tiedostoja. Välilehdellä näkyy myös projektin Git-repositorioon liittyviä tilastoja; päivitysten, haarojen, julkaisujen ja osanottajien määrät. Lehdellä on mahdollista tehdä myös ”pull request”, eli käyttäjä voi pyytää muita projektin osanottajia lataamaan omaan haaraansa tehdyt muutokset.

Issues-välilehdellä käyttäjä voi luoda kategorisoidun issuen, joka voi olla esimerkiksi ohjelmassa oleva virhe. Välilehdellä voi myös luoda virstanpylväitä, joille voi määrittää issueita. Virstanpylväät auttavat projektin etenemisen jaksottamisessa.

Pull requests –välilehdelle on listattu kaikki projektin aiemmin mainitut ”pull requestit”.



Kuvio 3. Testiprojektin etusivun välilehdet GitHub-järjestelmässä

Projects-välilehti simuloi Kanban-taulua. Käyttäjä voi luoda erillisiä tauluja tekemättömille, keskeneräisille ja tehdyille tehtäville. Esimerkiksi tekemättömien tehtävien taulun voi säätää ottamaan vastaan kaikki uudet issuet automaattisesti.

Wiki-välilehdellä käyttäjä voi selata projektiin liittyviä wiki-sivuja. GitHubin tukemilla formaateilla, kuten Markdownilla, RST:llä ja Textilellä, muotoillut sivut näytetään tällä sivulla HTML-muodossa.

Insights-välilehdellä käyttäjä voi tarkastella erilaisia projektiin liittyviä tilastoja, esimerkiksi projektin osanottajien aktiivisuutta, projektin kloonien määrää, päivitysten määrää sekä koodiin tehtyjen lisäyksien ja poistojen määrää. Tilastoja voi käyttää esimerkiksi kehitysprosessien tehostamiseen sekä projektin etenemisen valvontaan.

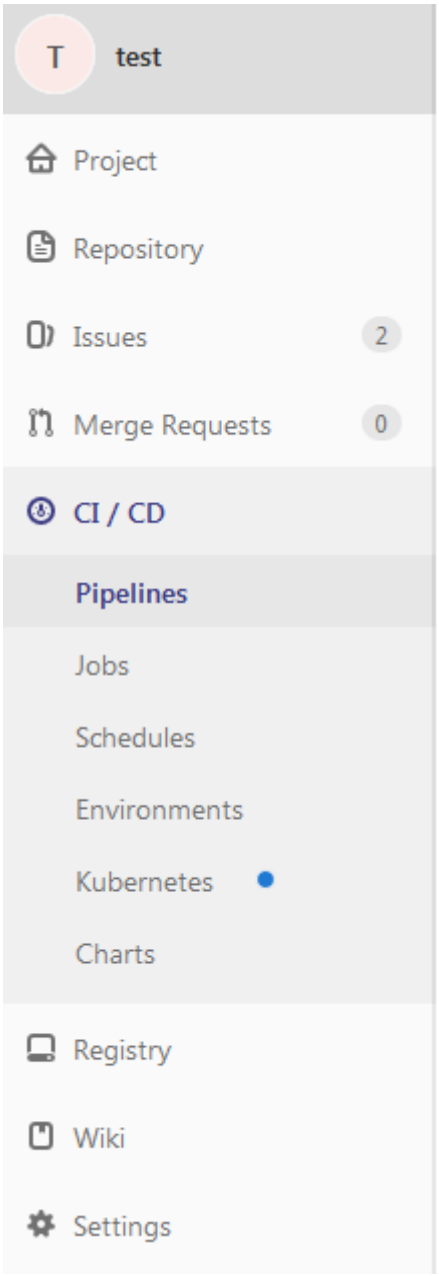
6.2 Gitlab

6.2.1 Lyhyesti

GitLab on pitkälti samanlainen palvelu kuin GitHub. Sen wiki-sivut tukevat Markdownia, RDocia ja AsciiDocia.

6.2.2 Projektisivu

Kuviossa 4 näkyvällä GitLabin projektin etusivulla on pitkälti samanlaiset välilehdet kuin GitHubissa. CI/CD-välilehti tarjoaa työkaluja jatkuvan integraation ja jatkuvan käyttöönoton tueksi. Jatkuvaa integraatiota voi toteuttaa repositorion juureen lisättävällä tiedostolla, jossa määritellään GitLabin runnerin toimintaohjeet. Oletuksena runneri ajaa pipelinen, jossa on kolme vaihetta; koonti, testaus ja käyttöönotto. Runnerit ovat virtuaalikoneita, joilla käännettyä koodia testataan.



Kuvio 4. Testiprojektin etusivun välilehdet GitLab-järjestelmässä

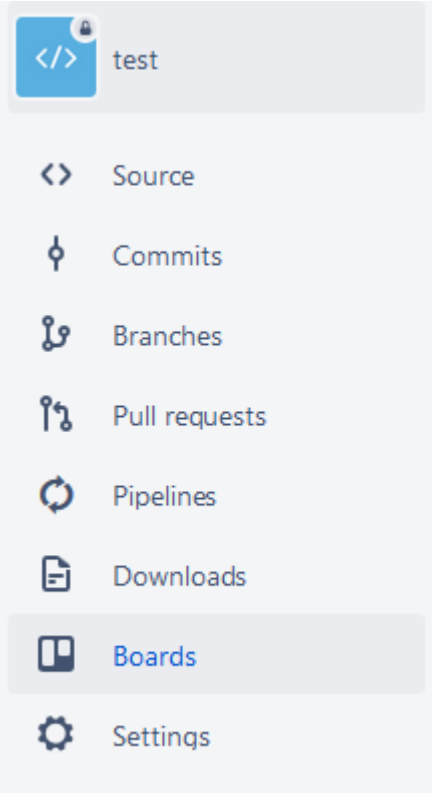
6.3 BitBucket

6.3.1 Lyhyesti

Bitbucket ei juuri poikkea GitLabista tai GitHubista muuten kuin siten, että se tukee Git-repositorioiden lisäksi Mercurial-repositorioita. Bitbucket käyttää wiki-sivuissaan Markdown-syntaksia.

6.3.2 Projektisivu

Kuviossa 5 näkyy Bitbucket-järjestelmässä olevan testiprojektin etusivu. Järjestelmässä on pipeline-ominaisuus, kuten GitLab-järjestelmässäkin. Boards-välilehdeltä voi aktivoida Trello-järjestelmän, joka on eräänlainen korttipöytä, jonka tarkoitus on helpottaa projektin organisointia.



Kuvio 5. Testiprojektin etusivun välilehdet Bitbucket-järjestelmässä

7 Siirto-operaatio

7.1 Aluksi

Toimeksiantaja rajasi Trac-järjestelmästä siirrettävät osat projektin repositorioon ja wiki-sivujen uusimpiin versioihin. Projektin issueiden siirtoa hän ei kokenut tarpeelliseksi. Siirto-operaatiota varten oli käytössä pääsy projektin Trac sivulle JAMK:n kirjautumistunnuksilla sekä projektin Subversion-repositorion osoite. Toimeksiantaja halusi, että työssä tutkitaan erilaisia siirtometodeja.

7.2 Siirtometodit

7.2.1 Repositorio

Repositorion siirtotapoja ei ollut kovin monia vaihtoehtoja. Projektin kohdeympäristön valinnan jälkeen varmistui, että Tracissa sijainnut Subversion-repositorio tulisi muuttaa Git-repositorioksi. Tähän tarkoitukseen löytyneitä työkaluja olivat Gitin oma svn-operaatio (git-svn) sekä subgit. Siirto-operaatiossa päädyttiin käyttämään Gitiin sisältyvää operaatiota, sillä Gitiä olisi tarvittu muutenkin.

Kuvioissa 6 ja 7 nähdään Subversion- ja Git-repositorioiden versionumerot. Numeroilla on merkitystä käytetyn dokumentaation valinnassa, sillä uudemmissa versioissa voi olla lisättyjä ominaisuuksia, joita ei vanhoista löydy.

Kuvio 6. Subversion-repositorion versio repositorion juuressa web-selaimella tarkasteltuna

Kuvio 7. Työaseman Git-sovelluksen versio

Kuviossa 8 näkyy osa Betty-projektin repositorion hakemistopuusta juuresta alkaen, eli src-hakemisto sijaitsee repositorion juurihakemistossa. Puussa on näytetty hakemistopuu Visual Studio -projektitiedostoon asti. Esimerkiksi Betty-hakemistossa se sijaitsi trunk-nimisen alihakemiston sisällä, ja kyseinen nimi viittaisi siihen, että jossain vaiheessa on pyritty toteuttamaan Subversionin suosittelemaa repositorion rakennetta. BettyInstallHelper-hakemiston tapauksessa projektitiedosto sijaitsi heti hakemiston sisällä. Tästä saattoi päätellä, ettei projektin aikana ole toteutettu minkäänlaista selkeää linjaa repositorion rakenteen suhteen. Repositoriossa ei ollut haaroja eikä tageja, mikä oikeastaan yksinkertaisti siirto-operaatiota.

```
\---src
    +---2011
    |   +---Betty
    |   |   \---trunk
    |   +---BettyInstallHelper
    |   +---BettyKomponentit
    |   |   \---trunk
```

Kuvio 8. Ote tiedostopuusta Betty-projektin Subversion-repositoriossa

Siirto-operaatiota edelsi kaikkien repositorioon päivityksiä tehneiden tietojen haku projektin repositoriosta Subversionin log-komennon avulla. Tämä tapahtui kuviossa 9 ilmenevällä tavalla, vain sillä erotuksella että tulostus ohjattiin tekstitiedostoon. Kyseinen komento tulostaa repositorion lokin, mikä sisältää kaikki repositorioon tehdyt päivitykset tietoineen. Tämä loki tulostettiin tiedostoon, mitä myöhemmin käytettiin repositorion muuntamisen yhteydessä.

```
$ svn log -q http://version.labranet.jamk.fi/Betty/ | awk -F '|' '/^r/ {sub("^ ", "", $2); sub(" $", "", $2); print $2" =
 "$2" <"$2">"}' | sort -u
e6413 = e6413 <e6413>
E6413 = E6413 <E6413>
salesa = salesa <salesa>
```

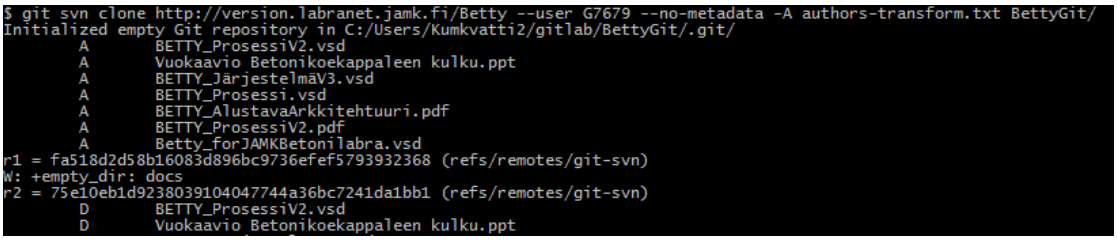
Kuvio 9. Käyttäjien etsintä Subversion-repositoriosta Git Bashin avulla

Repositorion tiedostoissa ei ollut juurikaan lisättyjä ominaisuuksia. Ainoat ominaisuustyypit uusimmassa revisiossa olivat kuviossa 10 näkyvät vs:project-root, svn:mime-type sekä svn:ignore. Näistä viimeisen saa siirrettyä Git-repositorioon repositoriomuutoksen jälkeen, eivätkä muut ominaisuudet ole säilyttämisen arvoisia; svn:mime-type kertoo tiedostotyypin ja vs:project-root on Visual Studion lisäämä ominaisuus, joka kertoo että kyseessä on Visual Studio -projektitiedosto. Lisäksi Gitissä ei edes ole mitään Subversionin ominaisuus-käsitettä vastaavaa toimintoa, joten näiden ominaisuuksien tallennus olisi saavutettuun hyötyyn nähden liian työlästä. Näin ollen repositoriossa tiedostoihin ja hakemistoihin sidotut ominaisuudet, eli metadata, voitiin jättää huomiotta repositorion muuttamisen yhteydessä.

```
$ svn proplist -R http://version.labranet.jamk.fi/Betty/ | awk '/  / {print $0}' | sort -u
  svn:ignore
  svn:mime-type
  vs:project-root
```

Kuvio 10. Lista erilaisista tiedostoihin ja hakemistoihin liitetyistä ominaisuuksista repositoriossa

Kuviossa 11 nähdään repositorion muuntamiseen Subversion-repositorisosta Git-repositorioksi tähtäävä operaatio. Kuviossa esiintyvä ”git svn clone” -komentoyhdistelmä luo paikallisen, tyhjän Git-repositorion, lataa Subverison-repositorion määritellystä polusta, muuttaa Subversionin revisionit Gitin commiteiksi ja lisää commitit paikalliseen Git-repositorioon. Kuviossa on myös nähtävissä Subversion-revisionumeroiden muuttaminen Gitin commit-tarkistussummiksi. Komentoyhdistelmä tulostaa myös kaikki Subversion-repositorioon tehdyt tiedostojen ja hakemistojen lisäykset, poistot sekä muokkaukset.



Kuvio 11. Subversion-repositorio kloonataan Git-repositorioksi BettyGit-hakemistoon

Kuviossa 12 nähdään listattuna juuri luodun Git-repositorion kaikki haarat. Haaroja on vain yksi, master-haara, ja se pitää sisällään koko Subversion-repositorion sisällön metadataa lukuun ottamatta.

Kuvio 12. Haarojen listaus muunto-operaation jälkeen

Kuviossa 13 siirretään Subversion-repositorion hakemistoissa ja tiedostoissa olleet svn:ignore-ominaisuudet .gitignore-tiedostoon. Kuviossa näkyvä varoitus ilmoittaa vain .gitignore-tiedoston End-Of-Line -merkin muuttumisesta, mikä ei kuitenkaan vaikuta Gitin tiedoston lukemiseen. Show-ignore -komento hakee kaikki kyseiset ominaisuudet rekursiivisesti.

```
$ git svn show-ignore | tee .gitignore

# /src/2011/BettyWeb/
/src/2011/BettyWeb/*.exclude

# /src/2011/BettyWeb/Error/
/src/2011/BettyWeb/Error/*.exclude

Kumkvatti2@Kumkvatti2-PC MINGW64 ~/gitlab/BettyGit (master)
$ git add .gitignore
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory.

Kumkvatti2@Kumkvatti2-PC MINGW64 ~/gitlab/BettyGit (master)
$ git commit -m 'svn:ignore-ominaisuudet siirretty .gitignore-tiedostoon'
[master 353d000] svn:ignore-ominaisuudet siirretty .gitignore-tiedostoon
 1 file changed, 6 insertions(+)
 create mode 100644 .gitignore
```

Kuvio 13. svn:ignore-ominaisuuksien siirto .gitignore-tiedostoon uuden Git-repositorion juureen

Kuviossa 14 asetetaan paikallisille repositolle origin-niminen etärepositorio. Näin voidaan ladata päivityksiä etärepositoriolta tai vastaavasti lähettää niitä sinne. Viimeiseksi siirretään kaikki paikallisen repositorion päivitykset etärepositorioon.

```
Kumkvatti2@Kumkvatti2-PC MINGW64 ~/gitlab/BettyGit (master)
$ git remote add origin https://gitlab.com/G7679/test4.git

Kumkvatti2@Kumkvatti2-PC MINGW64 ~/gitlab/BettyGit (master)
$ git remote -v
origin  https://gitlab.com/G7679/test4.git (fetch)
origin  https://gitlab.com/G7679/test4.git (push)

Kumkvatti2@Kumkvatti2-PC MINGW64 ~/gitlab/BettyGit (master)
$ git push origin master
Counting objects: 10841, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3852/3852), done.
Writing objects: 100% (10841/10841), 56.35 MiB | 518.00 KiB/s, done.
Total 10841 (delta 6786), reused 9638 (delta 5939)
To https://gitlab.com/G7679/test4.git
 * [new branch]      master -> master
```

Kuvio 14. Etärepositorion lisäys muutosoperaatiossa syntyneeseen Git-repositorioon

7.2.2 Wiki

Kuviossa 15 näkyvä alaviite oli liitteenä kaikissa wikisivuissa, ja kuviossa esiintyvä Plain Text -merkkijono on linkki, joka palauttaa koko html-muotoisen wikisivun sijaan varsinaisen wikisivun sisällön http-vastauksena.

Kuvio 15. Wikisivun alatunniste web-selaimella katsottuna

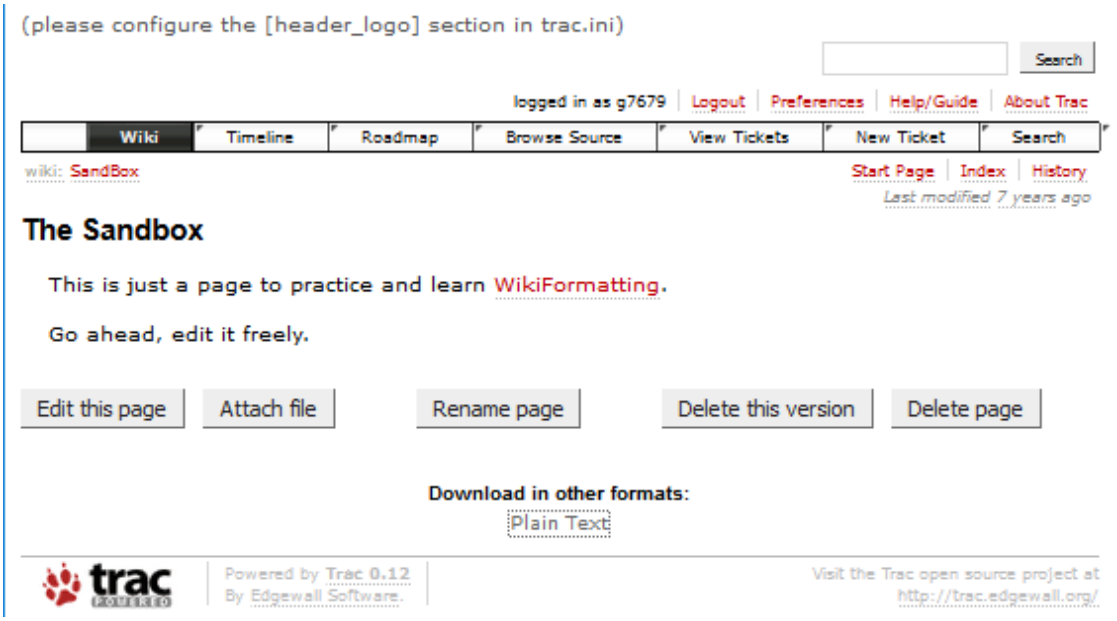
Trac-projektin wikiympäristöstä löytyi myös sivu, TitleIndex-sivu, johon kaikki wikisivut listautuivat automaattisesti aakkosjärjestykseen (Trac Macros 2014). Kyseistä sivua käytettiin olemassa olleiden wikisivujen nimien hakemiseen. Sivulla oli myös Tracissa valmiiksi olleet dokumentaatiosivut, jotka olisi voinut jättää siirron ulkopuolelle, mutta niiden nimeäminen oli kuitenkin niin epäsäännöllistä, että paremmaksi ratkaisuksi koettiin kaikkien sivujen siirtäminen.

Kuviossa 16 näkyvä div-elementti sisältyy jokaiseen Trac-järjestelmässä olevaan wikisivuun. Kyseinen elementti sisältää oleellista tietoa wikisivusta; sivun version, sivun version luojan sekä version luomisajan. Mikäli URL-osoitteen kyselyosassa ei ole määritelty versiota, näyttää Trac sivun uusimman version. Näin ollen saadaan kaikkien wikisivujen korkeimmat versionumerot lähettämällä http-pyynnöt wikisivuihin ilman kyselyosia, ja korkeinta versionumeroa käyttämällä saadaan ladattua kaikki wikisivun eri versiot alimmasta versiosta, eli versiosta 1, korkeimpaan.

```
<div class="trac-modifiedby">
    <a href="/trac/Betty/wiki/WikiStart?action=diff&amp;version=71"
       title="Version 71 by g7679">
       Last modified
    </a>
    <a class="timeline"
       href="/trac/Betty/timeline?from=2016-06-07T09%3A26%3A44%2B03%3A00&amp;precision=second"
       title="2016-06-07T09:26:44+03:00 in Timeline">
       6 months
    </a> ago
</div>
```

Kuvio 16. WikiStart-sivusta eroteltu div-elementti

Trac-järjestelmä tukee Wiki markup –kieltä, jolla muotoillut wiki-sivut se kääntää HTML-muotoon. Kuviossa 17 nähdään järjestelmän HTML-muotoon kääntämä Sand-Box-niminen wiki-sivu selaimella tarkasteltuna. Kuviossa 18 puolestaan nähdään sama wiki-sivu sellaisena kuin se on kirjoitettu.



Kuvio 17. SandBox-niminen wikisivu web-selaimella tarkasteltuna

```
[penne@piiras ~]$ curl -u g7679:$(cat .password) http://version.labranet.jamk.fi/trac/Betty/wiki/SandBox?format=txt
= The Sandbox =

This is just a page to practice and learn WikiFormatting.

Go ahead, edit it freely.[penne@piiras ~]$
```

Kuvio 18. SandBox-nimisen wikisivun http-vastaus käytettäessä format=txt -kyselyosaa

8 Lopputulos ja pohdintoja

Toimeksiantaja oli tyytyväinen lopputulokseen, vaikka siirto-operaation ohjelmallinen tarkastus jäi tekemättä. Manuaalinen tarkastus paljasti, että oleelliset tiedot säilyivät siirto-operaation aikana, vaikka osa Tracin käyttämään wiki-syntaksiin liittyvistä tiedoista katosikin siksi, ettei GitLabin Markdown-syntaksista löytynyt vastaavaa muotoilua. Tracin wiki-sivujen historiatiedot saatiin talteen ajankohtineen, ja siirrettiin GitLabin wiki-repositorioon. Siirto-operaatio olisi todennäköisesti sujunut helpommin Tracin sqlite-repositorion avulla, mutta wiki-syntaksin muuttaminen olisi silti ollut yhtä työlästä.

Subversion-repositorion muuttaminen Git-repositorioksi oli varsin helppoa git svn –työkalun avulla, ja muutoksen aikana katosi vain svn:mime-type- ja vs:project-root –ominaisuudet, sillä Git-repositoriossa ei ole vastaavaa ominaisuutta. Jatkokehitys on kuitenkin mahdollista ilman näitä ominaisuuksia.

Työ osoitti, että valmistumisen nopeuttamiseksi kannattaisi valita mahdollisimman paljon itseä kiinnostava aihe opinnäytetyöhön. Erilaisten ohjelmistoprojekteja tukevien työkalujen tunteminen on kuitenkin varmasti hyödyllistä työelämässä, ja varsinkin wiki-sivujen siirto-operaatio osoitti, ettei koodista aina tule kaunista ja helppolukuista.

Lähteet

Chacon, S. & Straub, B. 2018. Pro Git. 2. p. Viitattu 3.6.2018. https://github.com/progit/progit2/releases/download/2.1.64/progit.pdf.

Collins-Sussman, B., Fitzpatrick, B. W. & Pilato, C. M. 2011. Version Control with Subversion: For Subversion 1.7: (Compiled from r5707). Viitattu 2.6.2018. http://svnbook.red-bean.com/en/1.7/svn-book.pdf.

Hartmann, D. 2006. Interview: Jim Johnson of the Standish Group. Viitattu 1.6.2018. http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS.

JAMKin hallinto. N.d. Artikkeli Jyväskylän ammattikorkeakoulun sivustolla. Viitattu 30.9.2016. https://www.jamk.fi/fi/Tietoa-JAMKista/JAMKin-hallinto/.

Jyväskylän ammattikorkeakoulu. N.d. Jyväskylän ammattikorkeakoulun verkkosivut. Viitattu 30.9.2016. http://www.jamk.fi/fi/Etusivu/.

Leon, A. 2015. Software Configuration Management Handbook. 3. p. Artech House. Books24x7. http://common.books24x7.com.ezproxy.jamk.fi:2048/toc.aspx?bookid=112000.

Rakennustekniikan laboratorioympäristö. N.d. Artikkeli Jyväskylän ammattikorkeakoulun sivustolla. Viitattu 30.9.2016. http://www.jamk.fi/fi/palvelut/testaus-ja-analysointi/rakennuslaboratorio/.

Rossberg, J., Ehn, J. & Olausson, M. 2014. Beginning Application Lifecycle Management. Apress.

Sommerville, I. 2016. Software Engineering. 10. p. Harlow: Pearson Education Limited.

Subversion Testimonials. N.d. Apachen Subversion-repositoriossa sijaitseva lista Subversionia käyttävistä tahoista. Viitattu 3.6.2018. http://svn.apache.org/repos/asf/subversion/tags/1.2.0-rc2/www/testimonials.html.

Trac Macros. 2014. Trac-dokumentaatiossa oleva lista järjestelmän makroista. Viitattu 3.6.2018. https://trac.edgewall.org/wiki/0.12/WikiMacros.